DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

LOCALIZACIÓN Y MODELADO SIMULTÁNEOS EN ROS PARA LA PLATAFORMA ROBÓTICA MANFRED.

Autor: Azucena Fuentes Ríos

Tutor: Luis Pedraza Gómara

Tutor: Luis Moreno Lorente

LEGANÉS, MADRID

OCTUBRE 2011

Índice general

Lista de Figuras IV

1. Introducción **1**

1.1. Robótica. 1

1.2. Modelos de aprendizaje del entorno. 3

1.3. Objetivos 5

1.3.1. Objetivos generales 5

1.3.2. Objetivos de carácter técnico 6

1.4. Estructura del proyecto 7

2. Estado del arte **9**

2.1. Evolución de la robótica 9

2.1.1. Robótica móvil 12

2.2. Localización y modelado 13

2.2.1. Mapas de navegación y su aplicación al SLAM 15

2.2.2. Aproximaciones al problema de localización. 22

3. Localización y Modelado Simultáneo **34**

3.1. La formulación Bayesiana del SLAM 35

3.1.1. Principales algoritmos. 39

3.1.1.1. Filtro de Kalman. 39

3.1.1.2. Método de Monte Carlo 41

3.1.1.3. Filtro de partículas 44

4. Plataforma de desarrollo **50**

4.1. Sistema Operativo. Linux . . . 50

4.2. Herramientas de Programación. . . . 51

4.2.1. C++ . . . 51

4.2.2. OpenCV . . . 52

4.2.3. OpenSlam . . . 53

4.2.4. RADISH: The Robotics Data Set Repository . . . 54

4.3. Frameworks . . . 55

4.3.1. Carmen . . . 56

4.3.2. Player/Stage . . . 57

4.3.3. ROS . . . 60

4.4. Manfred . . . 64

4.4.1. Estructura del robot. . . . 65

4.4.2. Sistema sensorial. . . . 66

4.4.3. Sistema locomotor. . . . 68

4.4.4. Sistema de alimentación. . . . 68

4.4.5. Sistema manipulador LWR-UC3M-1. . . . 69

4.4.6. Sistema de procesamiento. . . . 70

5. ROS **71**

5.1. Introducción. . . . 71

5.2. Objetivo del diseño . . . 73

5.3. Conceptos Fundamentales . . . 75

5.3.1. Nodos . . . 75

5.3.2. Mensajes . . . 76

5.3.3. Topic . . . 77

5.3.4. Servicios . . . 78

5.3.5. Master . . . 78

5.4. Casos de Uso . . . 79

5.4.1. *Depuración de un sólo nodo* . . . 79

5.4.2. *Registro y Reproducción* . . . 79

5.4.3. *Subsistemas* . . . 80

5.4.4. *Proyectos de colaboración.* . . . 81
5.4.5. *Visualización y Monitorización* . . . 82
5.4.6. *Transformaciones* . . . 84
5.4.7. *Grabación de datos* . . . 84
5.5. Conclusión . . . 85

6. Desarrollo **86**
6.1. Simulación VS Realidad . . . 86
6.2. Implementación . . . 89
6.2.1. CoreSlam . . . 89
6.2.2. Slam_Gmapping . . . 93
6.2.2.1. Algoritmo Gmapping . . . 94
6.3. Implementación Manfred . . . 96

7. Resultados experimentales. **98**
7.1. Experimentos y resultados con simulación. . . . 98
7.2. Experimentos y resultados de robot real: Manfred. . . . 105
7.3. Comparativa entre los algoritmos. . . . 111

8. Conclusiones y trabajos futuros. **112**
8.1. Conclusiones. . . . 112
8.2. Trabajos futuros. . . . 113

A. Player/Stage **114**

B. Azumapping **116**

C. Launch **120**

D. Planificación. Diagrama de Gantt **123**

Bibliografía **127**

Índice de figuras

1.1. Tareas que debe resolver el robot para la adquisición de modelos precisos del entorno. 4

2.1. Aplicaciones de Robots I. 10
2.2. Aplicaciones de Robots II. 11
2.3. Cierre de bucle. [Konolige & Gutmann 1999]. 16
2.4. Mapa topológico. 18
2.5. Mapa de características. El entorno es definido por una serie de características estáticas, en este caso puntos, y estos son utilizados para calcular el movimiento del robot. 19
2.6. Mapa de ocupación de celdillas. 20
2.7. Mapa construido mediante un algoritmo de máxima probabilidad incremental [13]. 21
2.8. Representación de la trayectoria: odometría frente a estimación odométrica mediante láser. 24
2.9. Elipses de error en torno a la trayectoria estimada de un robot. 26
2.10. Diferentes modelos de triangulación: a)Ángulos absolutos de referencia de las balizas.b)Ángulos observados entre balizas.c)Distancias a las balizas.d)Ángulos y distancia a una baliza. 27
2.11. Sistema de posicionamiento global basado en 24 satélites. 29
2.12. Localización por GPS. 30
2.13. Interferencia entre sonares(Crosstalk). Puede ser directo(a) o indirecto(b). 31

2.14. Escáner láser 3D 32

3.1. Función $f(x)$ cuyo valor esperado será evaluado con respecto a una distribución $p(x)$. 42

4.1. Simulador de robots Stage 59
4.2. Evolución de ROS. 61
4.3. Visualización y reconocimiento de objetos. 62
4.4. Ejemplos de robots que usan ROS. 63
4.5. Robot MANFRED 64
4.6. Telémetro láser PLS-220 de la casa SICK 67

5.1. Configuración típica de una red ROS. 73
5.2. Cabecera IDL. 74
5.3. Comunicación ROS-topic. 77
5.4. Comunicación ROS-services. 78
5.5. Sistema de navegación. 80
5.6. Sistema de archivos. 82
5.7. Rviz.Herramienta de visualización desarrollada por ROS. 83
5.8. Resultado de ejecutar la instrucción rxgraph de ROS. 84

6.1. Mapa del laboratorio CAOR. Como datos de entrada recibe valores brutos de láser y odometría. 90
6.2. Resultado de la integración en el mapa de un escaneo láser. Se observa la "V" del agujero dibujado alrededor de cada impacto de láser. 91
6.3. Mapa generado por SLAM (Grisetti, Stachniss e Burgard 2006) 93
6.4. Relaciones y comunicaciones que se establecen entre los distintos nodos mientras se ejecuta el algoritmo Gmapping. 96
6.5. Árbol de transformadas generado en Manfred. 97

7.1. Entorno Original 99
7.2. Laboratorio. 100
7.3. Entorno Original 101
7.4. Habitación. 101

7.5. Entorno Original 102
7.6. Semi-Laberinto. 102
7.7. Entorno Original 103
7.8. Laboratorio Willow. 104
7.9. Diseño CAD de la tercera planta del edificio Betancourt de la Universidad Carlos III de Madrid. 105
7.10. Pasillo de la tercera planta del edificio Betancourt. 106
7.11. Pasillo de la tercera planta del edificio Betancourt. 106
7.12. Resultado CoreSlam.Pasillo de la tercera planta del edificio Betancourt. 107
7.13. Resultado Gmapping.Pasillo de la tercera planta del edificio Betancourt. 108
7.14. Resultado Gmapping.Pasillo de la tercera planta del edificio Betancourt. 109
7.15. Resultado Gmapping.Pasillo de la tercera planta del edifico Betancourt. 109
7.16. Resultado CoreSlam.Pasillo de la tercera planta del edificio Betancourt. 110

B.1. Mapa generado con el algoritmo azumapping. 117
B.2. Mapa generado con el algoritmo azumapping. 119

D.1. Planificación. Diagrama de Gantt. 126

Resumen

La robótica móvil tiene entre uno de sus principales desafíos dotar a los robots de autonomía. Se pretende con ello crear robots capaces de navegar de forma autónoma en terrenos inhóspitos y desconocidos por humanos, ofrecer soluciones robóticas a problemas tales como operaciones de búsqueda y rescate, así como de exploración tanto espacial como submarina.

El presente Proyecto Fin de Carrera proporciona una amplia visión, tanto teórica como experimental, del problema de la Localización y Modelado Simultáneo o SLAM ("Simultaneous Localization and Mapping") con robots móvil. Se busca resolver los problemas que plantea colocar un róbot móvil en un entorno y posición desconocidos, y que él mismo sea capaz de construir incrementalmente un mapa del entorno al mismo tiempo que utiliza dicho mapa para determinar su propia localización.

Nuestro principal objetivo, será crear mapas de ambientes simulados y reales a partir de los datos proporcionados por los propios sensores del robot. La mejor forma de probar las aplicaciones se consigue trabajando con robots y entornos reales, ya que las simulaciones se asemejan pero nunca son perfectas. A pesar de esto, trabajar con modelos simulados tiene una serie de ventajas puesto que nos permiten realizar pruebas y depurar aplicaciones de forma sencilla y práctica. Además, ofrece la posibilidad de modificar y utilizar diferentes entornos según nuestras necesidades y todo ello sin poner en peligro la integridad del Robot ni de los elementos del entorno.

Como se explicará en los capítulos 6, 7 y 8 del presente proyecto, para realizar los experimentos hemos implementado dos algoritmos: *CoreSlam* y *Gmapping*, basados en técnicas SLAM. Se finalizará el documento con las conclusiones a las que se han llegado y se propondrán posibles mejoras.

Abstract

Mobile robotics has among one of the main challenges to equip autonomous robots. The aim is to create robots that can navigate autonomously in inhospitable lands unknown, by the human being. Moreover, robotics offers solutions to problems such as search and rescue operations as well as in space as undersea exploration.

This Thesis provides a broad overview, both theoretical and experimental, and the problem of Simultaneous Localization and Mapping with mobile robots. It seeks to resolve problems like placing a mobile robot in an environmment and location unknown, and that it is able to incrementally build a map of the environment while using this map to determine its own location.

Our main goal, is to create maps of simulated and real environments from data provided by the own robotś sensors. The best way to test applications, is achieved by working with robots and real environments because the simulations are similar but they are never perfect. Despite this, working with simulated models has a number of advantages because it allows us to test and debug the application in a simple and practical way. Besides being able to modify or use different settings to suit our needs and all this without compromising the integrity of the robot or the environment elements.

As explained in chapters 6, 7 and 8 of this project, to perform the experiments we implemented two algorithms:CoreSlam and Gmapping, based on SLAM techniques. Document will be finalized with the conclusions that were reached and recommendations made for improvements.

Capítulo 1

Introducción

En este primer capítulo presentamos el contexto general en el cual se encuadra nuestro Proyecto Fin de Carrera, así como la importancia que puede tener su aplicación en el mundo de la robótica. Se describen los objetivos generales del proyecto y se resume la manera en que está organizado el documento.

1.1. Robótica.

La robótica es el arte de percibir y manipular a través de dispositivos controlados por computador, el mundo real. Los robots son capaces de captar datos del mundo a través de diferentes sensores (láser, sonar) y de ejecutar diferentes operaciones gracias a actuadores. Sus aplicaciones son muy diversas, existen dispositivos móviles dedicados a la exploración de planetas, robots diseñados para usos industriales, robots de utilización en cirugía asistida, etc.

Pese a que la robótica lleva estudiándose más de 25 años está aún en sus orígenes. A día de hoy pensar en robots que nos ayuden en nuestras tareas cotidianas es una fantasía, pero cada vez más próxima. Uno de los principales problemas es que no nos movemos en un mundo predecible, hay cosas que cambian rápidamente, y por ello hay que buscar que los robots sean capaces de detectar estas variaciones y puedan tomar un cambio de decisión.

El gran desafío al que se enfrenta la robótica móvil es la de conseguir máquinas verdaderamente autónomas; para llegar a lograrlo han de tenerse en cuenta diferentes aspectos. El

primer problema que se nos plantea es el de la *exploración*, debemos conocer el entorno para poder navegar a través de él, surgue así el segundo problema a resolver: la capacidad de navegación, que trataremos en el siguiente párrafo. La cuestión que afronta el robot móvil es realizar una exploración completa del entorno e ir construyendo un mapa global a medida que va inspeccionando nuevas partes del mismo. Cuando trabajemos en entornos desconocidos nos apoyaremos en la odometría [1] y en los datos obtenidos a través de sensores para ir conociendo dicho entorno.

Como avanzamos en el párrafo anterior, el segundo problema a resolver es la *capacidad de localización*, que debe permitir de forma autónoma que el robot alcance diferentes posiciones de acuerdo con alguna especificación proporcionada por sí mismo o por otra entidad. Por esta razón, la planificación de caminos es una de las capacidades deseables en una primera aproximación. En el caso de un entorno conocido y haciendo uso de la información sensorial recibida la resolución de esta clase de problemas consistirá en: la optimización del coste de viajar de una posición inicial hasta una final mientras se van evitando obstáculos, optimización del tiempo y velocidad de navegación, obtención de trayectorias suaves, etc.

Por último, se deberá tratar el problema de la *localización*. Una de las cuestiones principales para resolver este problema consiste en cómo hacer coincidir los datos que obtienen los sensores del robot (visión, sonar, láser, infrarrojos) con los datos de un mapa modelo del entorno. El problema de la localización puede resolverse mediante métricas garantizando que la diferencia entre la posición real del robot y la posición en la que el robot piensa que está es pequeña y limitada, o puede resolverse por medios cualitativos y topológicos. Cabe decir, que basar el posicionamiento del robot en la odometría no es una técnica eficaz puesto que conduce a errores ilimitados en la estimación de la posición como veremos en el capítulo 2.

Para llevar a cabo la localización existe un amplio repertorio de sensores, aunque el más difundido es el uso de escáneres láser de medición de distancia puesto que representan el entorno con mejor precisión (el error puede llegar a ser de $\pm$1mm) que por ejemplo los sensores tipo sonar. En el caso de que el robot estuviese dotado de cámaras de visión que nos permi-

[1] La odometría es un método sencillo que nos permite estimar la posición basándose en la información sobre la rotación de las ruedas para estimar cambios en la posición a lo largo del tiempo.

tiesen comparar las imágenes de la cámara con un modelo a priori del entorno, seguiríamos decantandonos por un sensor láser ya que el tiempo dedicado al procesamiento de datos es muchísmo menor.

Queda así planteado uno de los desafíos actuales de la robótica móvil, el SLAM (*Simultaneous Localization And Mapping*[2], Localización y Modelado Simultáneo de mapas), que se puede definir como una técnica usada por robots y vehículos autónomos para construir un mapa en un entorno desconocido mientras al mismo tiempo se mantiene actualizada la posición del robot dentro del mapa.

Esto no es tan sencillo como en un principio puediera parecer debido a; la dificultad de calcular el desplazamiento relativo del robot usando unos sensores que no tienen una precisión infinita y debido a que en el momento en que se cometa un error en la estimación de un desplazamiento relativo éste quedará acumulado, produciéndose una distorsióndistorsión del mapa que se está construyendo.

1.2. Modelos de aprendizaje del entorno.

En general, el aprendizaje de un mapa para un robot requiere la solución de tres tareas que son: el modelado, la localización y la planificación de una ruta.

- El **modelado** es el problema de la integración de la información obtenida a través de los sensores del robot en una representación dada. Puede ser definida por la pregunta "¿Cómo es el entorno que me rodea?" Aspectos centrales del modelado son la representación del entorno y la interpretación de los datos de los sensores.
- La **localización** es el problema de estimar la *pose*[3] del robot en relación con un mapa. El robot tiene que responder a la pregunta "¿Dónde estoy?". Por lo general, se distingue entre localización local donde la pose inicial es conocida, y localización global en el que no hay un conocimiento a priori de la posición de partida.

[2]En ocasiones se puede encontrar que este término se refiere a *Self-localization And Mapping*

[3]En robótica móvil, se habla generalmente de "pose" para referirse a la posición y orientación del robot conjuntamente, valores que determinan completamente la situación de la máquina en el espacio de 2 ó 3 dimnesiones.

- La **planificación de la ruta** consiste en cómo orientar de manera eficiente al robot hacia un lugar deseado a lo largo de una trayectoria. La pregunta que debemos plantear en este caso sería "¿Cómo puedo llegar a un lugar determinado?".

Por desgracia, estas tres tareas no pueden ser resueltas de forma independente las unas de las otras. Antes de que un robot pueda responder a la pregunta de a qué se parece el entorno dado un conjunto de observaciones, es necesario saber en qué lugares se han hecho estas observaciones. Al mismo tiempo, es difícil calcular la posición actual del robot sin un mapa. La planificación de una ruta hasta una meta dada también está estrechamente ligada con el conocimiento del entorno, así como con la información sobre la actual posición del robot.

El siguiente diagrama (Figura 1.1), muestra los tres problemas tratados: modelado, localización y planificación, así como los problemas combinados entre ellos que están representados por las zonas de solapamiento.

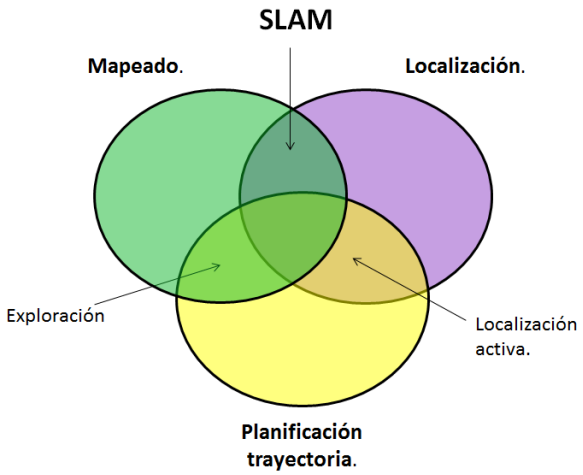


Figura 1.1: *Tareas que debe resolver el robot para la adquisición de modelos precisos del entorno.*

La localización y modelado simultáneo (SLAM) es el problema de la construcción de un mapa al mismo tiempo que el robot se localiza en ese mapa [32]. Ambas tareas no se pueden separar y resolver de manera independiente. Es por esto que, SLAM es conocido como el problema del huevo y la gallina: un buen mapa es necesario para una localización precisa, mientras que una estimación de la pose es necesaria para la construcción de un mapa. La zona central del

diagrama representa los enfoques integrados, que buscan soluciones simultáneas al problema de planificación, localización y modelado (SPLAM).

Debemos mencionar, que todos estos problemas se hacen aún más complejos en el caso de que el entorno cambie con el tiempo. La mayoría de las técnicas de mapeo asumen que el entorno es estático y no varía con el tiempo. Esto, sin embargo, es un supuesto poco realista ya que en la mayoría de los lugares donde los robots se utilizan a menudo hay gente caminando por el entorno, existen puertas que se abren y cierran o muebles que cambian de ubicación.

Finalizaremos el capítulo resumiendo los principales problemas a los que debe enfrentarse un robot a la hora de construir un mapa:

- Dónde guiar a un robot durante la *exploración*.[4]
- Cómo hacer frente al ruido generado por la pose y las observaciones.
- Cómo lidiar con la incertidumbre en el modelo del mundo obtenido por robot y la forma de interpretar los datos de los sensores.
- Cómo modelar los cambios en el entorno a través del tiempo.

1.3. Objetivos

A continuación, exponemos los objetivos que se pretenden alcanzar durante la realización de este proyecto. Se diferencia entre dos tipos de objetivos: los objetivos generales del proyecto y los de carácter técnico que se han planteado a la hora de llevarlo a la práctica.

1.3.1. Objetivos generales

Como ya se ha mencionado anteriormente el problema al que nos enfrentamos es el planteamiento, diseño e implementación de una solución que nos permita obtener la localización y construcción de mapas (SLAM) de un robot móvil utilizando como principal elemento sensor un es-

[4]Imporante diferenciar entre *mapping* consiste en generar correctamente un mapa del ambienete a partir de la informacón disponible y *exploración* que es la tarea de recorrer un ambiente desconocido con el fin de obtener los datos necesarios para su mapping.

cáner láser y trabajando tanto en entornos simulados como reales. Para lo cual, nos planteamos la siguiente serie de objetivos:

- **Análisis bibliográfico I** Lo primero será adquirir los conocimientos necesarios para abordar un problema de localización y construcción de mapas, es decir, conocer en profundidad el enunciado básico del problema de Localización y Modelado Simultáneo como uno de los pilares de un sistema robótico completamente autónomo.
- **Análisis bibliográfico II** Entender y formular los problemas de localización y construcción de mapas (SLAM) para robots móviles, aprendiendo el funcionamiento de los procedimientos de SLAM.
- **Implementación de técnicas de SLAM** Manejar e implementar diferentes algoritmos de generación de mapas analizando los procedimientos de adquisición y procesamiento de datos necesarios.
  No se pretenden desarrollar nuevas ténicas de SLAM sino implementar aquellas que se adecuen más al tipo de sensor utilizado y que tengan en cuenta diferentes restricciones del entorno.
- Conocer y utilizar herramientas de programación de sistemas robóticos como Player/Stage y ROS.
- Probar el funcionamiento sobre el simulador Player/Stage y sobre el robot real MANFRED.

1.3.2. Objetivos de carácter técnico

Con la realización de este proyecto se pretende adquirir cierta experiencia en el desarrollo de proyectos de ingeniería completando así la formación adquirida a lo largo de la carrera y profundizando en ciertos aspectos como son:

- Encontrar, analizar y afrontar los diversos problemas que surgen durante las distintas etapas del desarrollo.
- Realizar un primer acercamiento y conocimiento del mundo de la robótica.
- Comprender el funcionamiento y manejo de dispositivos robóticos.

- Iniciarse en el mundo de la investigación al tratar de desarrollar alternativas e ideas no encontradas en las publicaciones consultadas o de campos todavía en fase de desarrollo.
- Contrastar las diferentes alternativas disponibles para realizar la misma tarea, evaluar y seleccionar la mejor alternativa para un caso concreto.
- Mejorar el conocimiento ya adquirido en ciertas áreas de la informática como son el lenguaje C, C++ y los entornos y bibliotecas de libre distribución como Linux, Player/Stage, ROS, etc, observando las ventajas e inconvenientes (falta de documentación, software incompleto o con fallos...) que presentan.

1.4. Estructura del proyecto

La estructura de este documento se divide en una serie de capítulos bien diferenciados para facilitar el entendimiento y seguimiento del trabajo realizado a lo largo del proyecto. A continuación se muestra una pequeña descripción de cada uno de ellos.

- **Capulo 1 : Introducción.** Este capítulo tiene un carácter introductorio y con él se pretende guiar al lector en la problemática que aborda el proyecto. En esta parte del documento se hace una descripción del problema que se pretende resolver y se establecen los objetivos que se quieren conseguir.
- **Capítulo 2 : Estado del arte.** En este capítulo se hace un breve prolegómeno acerca de la historia de la robótica y se realiza un repaso del actual estado del arte en el campo de la Localización y Modelado Simultáneos. Se muestran las principales estrategias empleadas para modelar entornos explorados mediante robots móviles de entre las que destacan aquellas basadas en técnicas probabilísticas. Se consigue de este modo dar cabida y modelar en la medida de lo posible, no sólo el propio entorno, sino también las incertidumbres y errores presentes en cualquier medida adquirida por los sensores disponibles a dia de hoy.
- **Capitulo 3 : Localización y Modelado Simultáneo. SLAM** En este apartado se presentan las bases matemáticas del problema de la Localización y Modelado Simultáneo de robots móviles. Se indican las expresiones matemáticas que modelan el problema del SLAM y se introducen los conceptos básicos de los algoritmos existentes.

- **Capitulo 4 : Plataforma de desarrollo.** En este capítulo se describen los elementos empleados en el desarrollo del proyecto, tanto hardware como software. Además se presenta a Manfred, el robot móvil autónomo antropomórfico desarrollado por el departamento de Ingeniería de Sistemas y Automática de la Universidad Carlos III de Madrid con el que se ha trabajado en parte del proyecto.

- **Capitulo 5 : ROS** En esta sección llevamos a cabo una detallada descripción de la herramienta ROS, las aplicaciones que pueden tener y algunas de las librerías que nos ofrece.

- **Capitulo 6 : Desarrollo** En este capítulo se describen los algoritmos empleados desde un punto de vista funcional y se explica como se ha llevado a cabo la implementación.

- **Capitulo 7 : Experimentos y Resultados** Dentro de este capítulo se comentarán los experimentos llevados a cabo junto con los resultados obtenidos y se mostrarán visualizaciones del desarrollo de los mismos. Los resultados incluyen tanto experimentos con datos simulados, como experimentos con datos extraídos de entornos reales.

- **Capitulo 8 : Conclusiones y Líneas de trabajo futuras** En este apartado se hace una valoración global del trabajo realizado, exponiendo las conclusiones obtenidas tras la realización del proyecto, y se apuntan algunas posibles tareas a desarrollar en un futuro.

Capítulo 2

Estado del arte

Comenzaremos este capítulo con una breve introducción sobre la historia de la robótica, su evolución a lo largo del tiempo y algunas de sus aplicaciones típicas. Seguidamente explicaremos los distintos tipos de mapas que podemos utilizar para representar el entorno, y entenderemos porque es uno de los elementos clave para que un robot pueda localizarse. Cerraremos esta sección abordando el problema de la localización de sistemas móviles, sus tipos, dificultades y recursos necesarios para llevarlos a cabo.

2.1. Evolución de la robótica

El hombre siempre se ha apoyado en la ciencia para buscar una vida más fácil, fabricando herramientas y máquinas que le permitiesen descargarse del trabajo. Estas máquinas han ido evolucionado poco a poco hasta llegar a lo que hoy se conoce como **robot**.

El término robot tiene su origen en la obra checoslovaca Rossum's Universal, del autor Karel Capck, publicada en 1921. En esta obra el término robot surge como derivado de la palabra checa *robota* que significa trabajo forzado o servidumbre, y que se usó para nombrar a unas máquinas construidas por el hombre, dotadas de inteligencia, que se ocupaban de los trabajos pesados.

Durante el siglo pasado la robótica sufrió un gran despliegue, sobre todo con la aparición de nuevas ciencias como la electrónica, informática o la inteligencia artificial. Los robots son

cada vez más inteligentes y autónomos tendiéndose hacia una sociedad crecientemente robotizada. Hoy en día las grandes industrias se apoyan en el uso de robots para tareas complejas que requieren gran precisión (Figura 2.1(a)); campos como la medicina también hacen uso de la robótica aumentando la interacción de estos con el ser humano (Figura 2.1(b)).

(a) Robot soldando (b) Robot enfermero creado por Mutsubishi

Figura 2.1: *Aplicaciones de Robots I.*

Las empresas construyen robots de todo tipo, unos destinados a realizar tareas domésticas (Figura 2.2(a)), otros destinados a la diversión como la mascota Aibo de SONY (Figura 2.2(b)), incluso androides o humanoides capaces de desarrollar comportamientos inteligentes. Algunos ejemplos de los robots más avanzados son los creados por la NASA utilizados para la exploración del espacio (Figura 2.2(c)).

A pesar de que el mercado de robots industriales está muy maduro, no ocurre lo mismo con la robótica de servicio siendo éste muy pequeño, a excepción de Japón. Esta situación se debe en gran medida al hecho de que el consumidor se muestra reacio a utilizar en su vida cotidiana estas nuevas tecnologías, a lo que se une el elevado precio que adquieren estas máquinas.

Uno de los principales problemas a los que debe enfrentarse la industria de los robots autónomos es la complejidad de crear nuevos productos, lo que conlleva a la necesidad de no empezar de cero. A esto hay que sumarle el hecho de que la mayor parte de la investigación es privada y carece de un ecosistema estable de proveedores. Se puede decir, que es una indus-

(a) Robot tareas

(b) Mascota Aibo

(c) La sonda SPIRIT

Figura 2.2: *Aplicaciones de Robots II.*

tria con un alto riesgo debido a las elevadas inversiones que realiza y al poco retorno percibido, además, como ya hemos comentado, sigue habiendo una gran resistencia al cambio que les lleva a preguntarse si ¿todavía es demasiado pronto?

Un aspecto que resulta interesante, es la importancia que está adquiriendo en la actualidad la investigación académica la cual está muy adelantada respecto a la industria y al mercado. A pesar de ésto, es llamativo el poco feedback que existe entre la Universiad y la Empresa que hace que sólo se llegue a soluciones teóricas y simuladas quedándose en pruebas de concepto, y en la consiguiente pérdida de tiempo e infraestructuras. Otro de los problemas a los que debe enfrentarse, es la imposibilidad de reutilización ya que falta colaboración entre las partes.

Centrándonos en el contexto de un proyecto, los problemas fundamentales a los que se enfrenta son el tamaño del mismo y el conocimiento necesario para abordarlo. Las necesidades de proyectos medios o grandes son diversas entre ellas encontramos :

- Explosión de requerimientos software.
    1. Ubicuidad: la capacidad de estar presente en diferentes lugares y aplicaciones.
    2. Herramientas: desarrollo, configuración, depuración.
    3. Arquitecturas más robustas y flexibles.
- Procedimientos de ingeniería del software.
- Trabajo en equipo.
- Reutilización del software.
- Reutilización del conocimiento.

Todo lo anterior ha tenido como consecuencia el desarrollo de una Robótica cada vez más distribuida que propone una solución orientada a los servicios y componentes, defendiendo una Programación de propósito general:

1. TCP/IP, CORBA, ACE, etc.
   Surguiendo los Robotic Software Frameworks (RSFs).
2. Player, Orca, YARP, OROCOS, CARMEN, JADE, etc.

Algunos de estos RSFs han sido empleados en este proyecto y serán desarrollados con más detalle en el capítulo 4. Todas estas novedades mejoran la situación, pero se necesitan más mejoras.

2.1.1. Robótica móvil

La robótica móvil estudia a robots que realizan tareas desplazándose autónomamente en entornos dinámicos y complejos, es decir, desde espacios interiores como oficinas y casas, hasta exteriores como espacios terrestres, submarinos y sustentados sobre el aire. La localización de un robot móvil en un entorno conocido es uno de los problemas fundamentales de la robótica móvil, junto con la construcción automática de mapas del entorno.

Inicialmente los investigadores daban al robot una representación del entorno, estos mapas eran hechos de manera externa al robot y por lo mismo eran difíciles de implementar con la perspectiva del robot (nuestra perspectiva es distinta a la del robot). Sin embargo, el objetivo actual es que el robot genere una representación del entorno utilizando sus propios sensores, teniendo en cuenta sus limitaciones intrínsecas y alcances.

Si nuestro fin es la autonomía, la habilidad de un robot para construir mapas consistentes es un requisito clave, en la sección 2.2.1 se explicarán los distintos tipos de mapas que pueden ser utilizados. Con el mapa del entorno, el robot podrá realizar tareas de localización y planeación de movimientos. La localización ofrecerá el punto de partida, la tarea a realizar será su meta y el cálculo de los movimientos necesarios se hará a partir de las restricciones expuestas en el mapa.

Se han propuesto un gran número de modelos, enfoques y técnicas para resolver ambos problemas. Los elementos fundamentales de la formulación de ambos son la definición de un *modelo de observación* que proporcione la probabilidad de las lecturas de los senores dada una posición en el entorno definido y la definición de un *modelo de movimiento* que proporcione la probabilidad de la siguiente posición del robot, dada la posición actual y la acción ejecutada. Se usan con frecuencia soluciones probabilísticas ya que dan soluciones más fiables, reduciendo la incetidumbre y permiten tratar el problema de manera eficaz.

2.2. Localización y modelado

Para navegar de forma robusta, un robot debe saber dónde se encuentra dentro de su entorno. En el contexto de los robots móviles, el problema general de la localización puede ser formulado de la siguiente forma.
Dado: Un modelo del entorno, como una descripición geométrica, un mapa topológico o una rejilla de ocupación (consultar sección 2.2.1).

Tarea: Estimar la localización del robot en el modelo basándose únicamente en observaciones efectuadas por el robot. Dichas observaciones suelen consisitir en información de odometría acerca de los movimientos realizados por el robot, y en información de distancias a los obstáculos cercanas obtenidas mediante sensores de alcance (sonar, láser[1]).

Un problema muy ligado al de localización es el de la construcción automática del mapa del entorno. Este problema se puede formular de la siguiente forma:
Dado: Una serie de observaciones realizadas por el robot evolucionando por el entorno. (Las observaciones, al igual que en el problema de localización, suelen consistir en información de odometría e información de lecturas de alcance.)
Tarea: Construir un modelo del entorno que pueda ser utilizado por los algoritmos de localización. Posibles modelos son descripciones geométricas del entorno, mapas topológicos o rejillas de ocupación.

Los modelos y métodos propuestos para resolver los problemas de localización y modelado deben tener en cuenta una serie de limitaciones y restricciones prácticas del funcionamiento de los robots móviles. Algunas de ellas son las siguientes:

1. **Funcionalidad de los sensores:** El rango de percepción de los sesores (sonar, láser, cámaras) está limitado a una zona pequeña alrededor del robot. Para adquirir información global, el robot debe explorar activamente su entorno.

2. **Ruido en los sensores:** Las observaciones realizadas por los sensores son ruidosas y su distribución estadística no suele ser sencilla de modelar.

3. **Ruido en la posición:** Los movimientos del robot no suelen ser exactos, produciéndose errores de odometría. Estos errores son acumulativos con la distancia recorrida. Pequeños errores en la rotación del robot pueden tener efectos importantes en la estimación de los movimientos de traslación y en la determinación de la posición del robot (consultar sección 2.2.2).

[1]Es importante resaltar que sólo se dispone de información local sobre la posición del robot. Si se contara con información global sobre su posisción (mediante un sistema GPS, por ejemplo), el problema de la localización estaría resuelto

4. **Entorno complejos y dinámicos:** Los entornos en los que se mueve el robot suelen ser complejos y dinámicos, siendo complicado mantener modelos consistentes de los mismos.

5. **Necesidad de tiempo real:** Los requisitos de la aplicación (control de un robot móvil) obligan a procesar la información en tiempo real. Esto limita la complejidad del procesamiento realizado por los métodos de localización, así como los modelos del entorno.

2.2.1. Mapas de navegación y su aplicación al SLAM

Como hemos comentado con anterioridad un elemento fundamental de la localización y el mapeado es el modelo de representación del entorno. Un modelo o mapa del entorno es una abstracción con la que se representan únicamente aquellas caracteristicas del entorno que se consideran útiles para la navegación o localización del robot.

La utilidad principal de un modelo del entorno es proporcionar el elemento fundamental para la localización del robot. En general, los algoritmos de localización suelen comparar las lecturas obtenidas por los sensores del robot con el modelo del entorno, actualizando lal posición del robot de forma acorde con el resultado de esta comparación. La forma de realizar la comparación depende del tipo de modelo del entorno y de la propuesta realizada.

A continuación explicaremos los tipos de mapas usados con más frecuencia en sistemas de localización, para cada uno de ellos haremos una breve descripción funcional y estudiaremos si sirven para la localización con un mapa a priori. Los criterios que utilizaremos serán:

- Representación de la incertidumbre: Los sensores del robot no pueden medir con exactitud el entorno, luego existirá un cierto grado de incertidumbre en la medida. Al mismo tiempo, la localización del robot se deriva de este mapa, luego su pose estimada también tendrá incertidumbre. Por todo esto, un modelo de incertidumbre tendrá que reflejar con fidelidad el error entre el estado actual y el estimado del sistema.

- Convergencia monótona El primer objetivo de la incertidumbre es asegurar la convergencia del mapa. Un mapa es convergente si la geometría espacial del entorno se aproxima a la geometrá real cada vez que nuevas observaciones son incorporadas, es decir, se con-

sigue ir reduciendo la incertidumbre del mapa. Sin esta medida de la incertidumbre, un objeto estático con una localización estimada (x1,y1) puede ser desplazado con las actualizaciones del mapa a una posición distinta (x2,y2).

- Asociación de datos La representación del mapa tiene que ajustarse lo más posible a la realidad, para ello se compara la información obtenida de los sensores del robot y la información almacenada en el mapa. Es importante que la exploración del entorno sea lo suficientemente eficiente para operar en tiempo real, por otro lado es aconsejable que la asociación sea robusta para asegurar el buen funcionamiento en caso de trabajar en espacios grandes o en situaciones en las que no es necesario explorar todo el entorno pues una parte de él ya ha sido previamente modelado y sólo existe una pequeña región sin explorar.

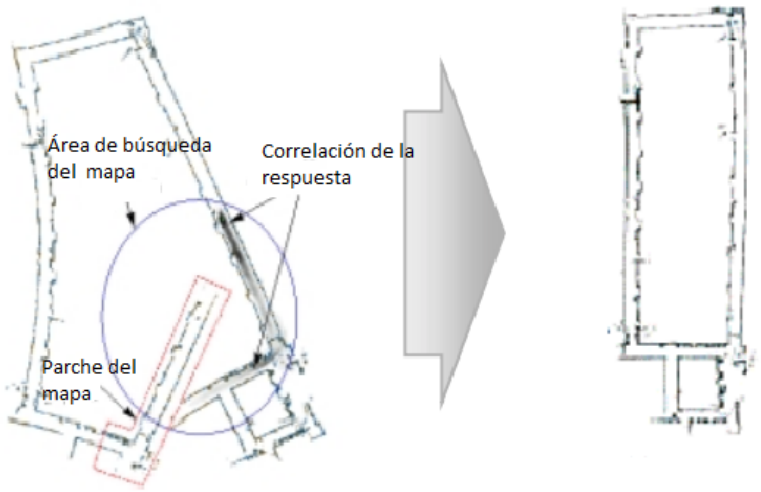


Figura 2.3: *Cierre de bucle. [Konolige & Gutmann 1999].*

- Detección de un ciclo Nos referimos a la detección de ciclo, cuando un robot que está realizando la exploración de un entorno es capaz de identificar que vuelve a una región del mapa ya explorada y que por tanto ha cerrado el bucle de su trayectoria(Figura 2.3). Este aspecto está muy ligado al concepto descrito en el párrafo anterior de la asociación de datos. La eficiencia de búsqueda es un aspecto importante pero aún más lo es la robustez en la decisión de si una asociación es correcta o no. Una vez encontrada una asociación correcta, el error acumulado en el mapa tiene que ser compensado correctamente cada vez que se actualice el mapa.

- Tiempo de computación almacenado El mapa tiene que almacenar suficiente información para permitir asociación de datos y convergencia. Este almacenaje y el tiempo de computación tienen que ser proporcionales al área tratada.

1. Mapas topológicos

Los mapas topológicos representan las características del entorno por el robot móvil usando un grafo. Los nodos representan lugares distintivos del entorno, constituyendo la unidad elemental de localización, y los arcos caminos a ser recorridos por el robot entre los lugares. Los lugares representativos del entorno (landmarks) son seleccionados en base a sus características distinguibles de forma unívoca.

El SLAM topológico explora el entorno siguiendo una serie de criterios con los que determina la trayectoria, al mismo tiempo va guardando las descripciones de los lugares. Cada vez que un lugar nuevo es encontrado, es conectado al lugar anterior siguiendo las especificaciones hasta llegar a él. De esta manera, el mapa es construido como secuencia lineal de lugares que continúa hasta que un lugar encontrado cuadre con uno anteriormente almacenado. Si este lugar puede ser catalogado como uno anterior entonces se cierra el ciclo o bucle.

Uno de los problemas más importantes que presenta esta técnica es la asociación de datos. En caso de detección de un ciclo en un lugar que coincida con uno anteriormente almacenado, la asociación de datos pasa a ser ambigua y el lugar observado puede ser uno de los previamente registrados o uno nuevo que se asemeje, distinguir entre estos dos casos no siempre es fácil. Una posible solución sería introducir algún tipo de información métrica.

Un inconveniente de los mapas topológicos es que no se pueden representar zonas abiertas (habitaciones grandes, halls), debido a que el alcance limitado de los sensores no obtiene información del entorno. Otro inconveniente es que los mapas construidos dependen excesivamente de la historia de las percepciones del robot al construirlos.

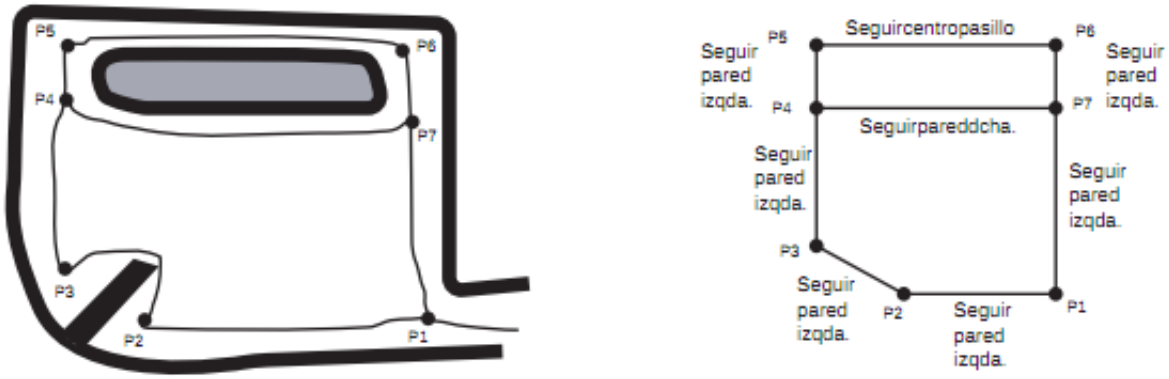


Figura 2.4: *Mapa topológico.*

Siguiendo el ejemplo de la figura 2.4, el arco entre P1 y P2 está etiquetado *seguir pared izquierda* porque esa es la conducta local que el robot siguió para ir de un nodo a otro. Esto hace que los mapas topológicos sean muy sensibles a la aparición de elementos no modelados (personas, obstáculos imprevistos) que proporcionan información sensorial muy distinta de la modelada, haciendo que el robot pierda su localización.

Por otro lado, los mapas topológicos proporcionan la ventaja a la hora de realizar una planificación de la trayectoria del robot, tales como facilitar la interfaz con planificadores simbólicos, proporcionar un interfaz más natural para la interacción con instrucciones humanas, permitiendo dar órdenes del tipo "*ir a la habitación A*"

2. Mapa de características

Los mapas de características representan el entorno mediante la localización global de ciertas característcas paramétricas tales como puntos o líneas, como podemos ver en la figura 2.5. La localización se lleva a cabo extrayendo primero características de los datos observados y asociándolos a características en el mapa. Las diferencias entre las características predichas y los lugares observados son usados para calcular la pose del robot.

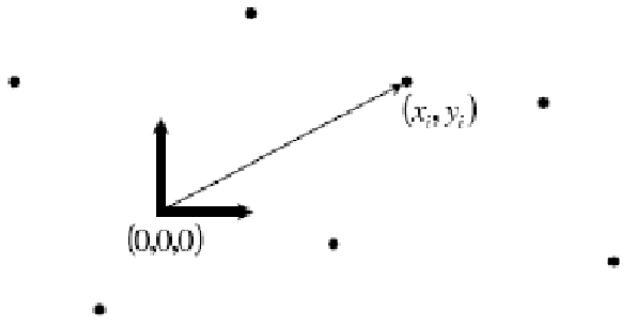


Figura 2.5: *Mapa de características. El entorno es definido por una serie de características estáticas, en este caso puntos, y estos son utilizados para calcular el movimiento del robot.*

Los puntos característicos en un mapa de características, a priori se asume que son perfectamente conocidos y cada uno de ellos es definido por sus correspondientes parámetros. Este método es bastante eficiente, además el espacio libre no es representado y por ello no añade coste de computación adicional al proceso de localización. El inconveniente que presenta es que no favorecen la planificación de rutas y tareas que tiene que ser abordadas como un problema diferente.

Uno de los problemas que presenta este tipo de mapas hace referencia a la asociación de datos. La correcta estimación de la pose depende de encontrar una correcta correspondencia entre el punto característico observado y el asociado al mapa. Una asociación errónea daría como resultado una reducción de la incertidumbre sobre la pose del robot y el error estimado aumentaría. Cabe decir, que estos mapas son sólo recomendables para entornos donde los objetos pueden ser fácilmente descritos por un modelo geométrico.

3. Mapa de ocuapción de celdilla

El algoritmo de ocupación de celdillas (*Occupancy Grid Mapping*) fue introducido por los trabajos de Hans Moravec [20] y Alberto ELfes [10] a mediados de la década de 1980. Se basa en discretizar el espacio, dividiéndolo en unidades de tamaño predefinido, que se clasifican como ocupadas o vacías con un determinado nivel de confianza o probabilidad, se les asigna un valor entre (0,1). La localización se hace comparando los datos de cada observación con el mapa usando técnicas de correlación cruzada(Figura 2.6).

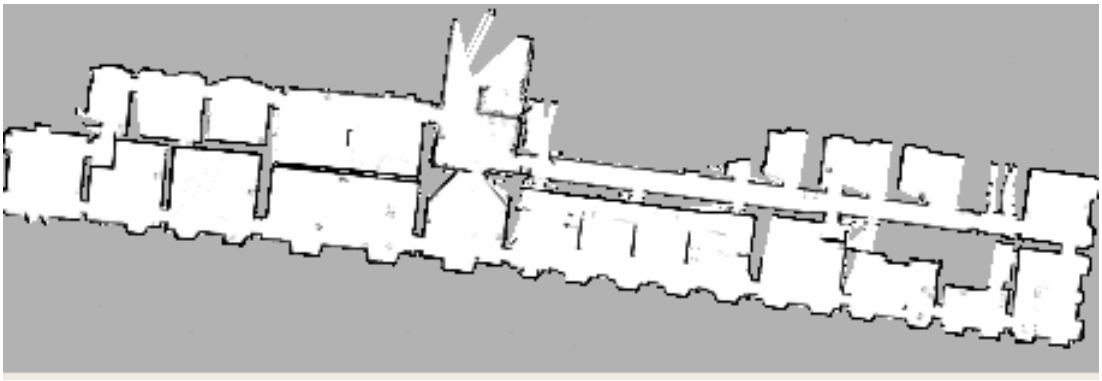


Figura 2.6: *Mapa de ocupación de celdillas.*

La técnica de Occupancy grid SLAM intercala los pasos de localización y actualización del mapa, primero registrando el mapa a corto plazo y luego fundiéndolo con el mapa global (localización), para más tarde actualizar la ocupación percibida del mapa global (mapeo). Este método funciona muy bien para entornos dinámicos y pequeños en periodos breves de tiempo, sin embargo no ofrece un modelo exacto de incertidumbre por lo que en periodos muy largos tiende a diverger [1].

Algunas de las ventajas que ofrece esta solución es que se trata de un algoritmo robusto y de implementación sencilla, además es capaz de distinguir entre zonas ocupadas y vacías, consiguiendo una partición y descripción completa del espacio explorado. Por este motivo es popular en tareas de navegación, al facilitar la planificación y generación de trayectorias empleando métodos convencionales.

En cuanto a las desventajas, decir que presenta dificultades a la hora de asociar datos lo que origina que su capacidad para cerrar bucles correctamente no se satisfactoria. Aunque la mayor dificultad en este tipo de mapas, sobre todo en lo que concierne a espacios grandes, es la relación entre la resolución de la malla y el tiempo computacional.

Lo ideal sería hacer la malla lo más pequeña posible, pero ello conllevaría un aumento del tiempo de computación. Por contra, si se eligen celdas demasiado grandes, se perderá información. El tiempo de computación también puede ser grande en tareas de planificación de

rutas si el mallado es pequeño. Existen métodos de tamaño de malla variable que permiten centrarse en las zonas de interés del mapa, pero tienen sus propias dificultades y son difíciles de implementar.

4. Máxima probabilidad incremental

Los buenos resultados obtenidos con los mapas de celdilla, así como su necesidad de disponer de una estimación para la posición del robot (aunque esta no fuera modelada probabilísticamente), animaron a los investigadores a explorar técnicas más simplificadas. Así, la idea básica de los algoritmos de máxima probabilidad incremental (*Incremental ML*)[35] es la de obviar toda noción de incertidumbre asociada a los elementos del mapa. Simplemente optan por mantener la estimación del mapa m* y de la posición del robot s_t^* más probable en cada instante, maximizando la siguiente expresión:

$$\{s_t^* , m^*\} = argmax_{s_t ,m} p(z_t \mid s_t , m) p(s_t , m \mid u_t , s_{t-1}^* , m^*) \tag{2.1}$$

La anterior maximización se suele limitar al cálculo de la posición del robot más probable a la vista de las medidas sensoriales obtenidas y al modelo no estocástico del mapa disponible.

A esta clase de algoritmos pertenecen las técnicas de *scan matching*[16][19][8], que se limitan a calcular la posición más probable del robot como resultado de alinear medidas brutas adquiridas por un sensor del láser, y construir incrementalemente el mapa sobre esta localización.

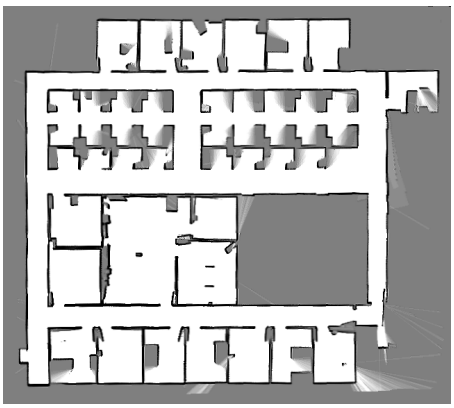


Figura 2.7: *Mapa construido mediante un algoritmo de máxima probabilidad incremental [13].*

Su principal ventaja es su sencillez y su ligereza computacional, pero encuentra serios problemas a la hora de cerrar bucles exitosamente, puesto que no mantiene ninguna información acerca de las correlaciones entre los diferentes objetos(Figura 2.7).

2.2.2. Aproximaciones al problema de localización.

La localización consiste en estimar la posición de un robot móvil en un mapa respecto a un sistema global de coordenadas. La información usada para resolver el problema de la localización proviene del entorno que rodea al robot (distancias a obstáculos próximos, identificación de patrones, marcas, etc) y de los sensores internos de éste, tales como, odómetros, brújulas, acelerómetros, etc, que no tienen conocimiento alguno del entorno del robot.

Los métodos de localización existentes pueden clasificarse en distintos grupos en función del criterio de selección escogido. Las clasificaciones empleadas no son excluyentes entre sí, pudiendo aparecer cada método en distintos grupos.

- Se pueden clasificar en dos grupos atendiendo a si la posición del robot se estima sin interpretar información del entorno o extrayendo información de él. Dentro del primer grupo se pueden incluir métodos como localización odométrica, localización mediante sistemas de navegación inercial, GPS, localización por estaciones de transmisión, estimación por balizas, etc. Dentro del segundo grupo se incluyen aquellos métodos que emplean sensores (telémetros láser, ultrasonidos,etc), para extraer características del entorno (esquinas, paredes...), obtener distancias a obstáculos, reconocimiento de patrones, entre otros muchos. Algunos de estos sitemas de obtención de información serán explicados más adelante.

- Otra clasificación alternativa divide los métodos de localización atendiendo a si estos proporcionan una localización local o global del dispositivo móvil. La *localización local*, llamada también re-localización, trata de mantener un seguimiento de la posición del robot, y para ello, estos métodos requieren conocer la posición inicial de éste. La mayoría de los algoritmos existentes pertenecen a este grupo, podemos poner como ejemplos: localización mediante marcas (naturales o artificiales), emparejamiento de características (matching), filtro de Kalman, etc. En cambio, la *localización global* trata de determinar la

posición del robot sin conocer la posición inicial que ocupa éste. Son métodos más robustos que los anteriores debido a la posibilidad de recuperar la localización del robot a partir de posiciones falsas. Algunos de los métodos de localización global más importantes son los Filtros de Kalman multi-hipótesis, los filtros basados en rejillas y los Filtros de partículas (también conocido como Método de Monte Carlo). En el capítulo 3 hablaremos con más detalle de estos métodos.

- Es posible hacer una última división diferenciando entre métodos probabilísticos y métodos no probabilísticos. Los segundos estiman la posición del robot sin tener en cuenta la incertidumbre del movimiento ni las características estadísticas de las observaciones. En cambio, los métodos probabilísticos sí que tienen en cuenta los errores producidos por los sensores internos y externos. Tratan de estimar la posición más probable del robot sobre un mapa del entorno a partir de las lecturas proporcionadas por los sensores. Trabajan en dos fases: una primera fase en la que el robot obtiene lecturas del entorno a partir de los sensores, estas lecturas son usadas para encontrar aquellas áreas donde es más probable que se encuentre el robot. En la segunda fase, el robot ejecuta un comando de acción, que le permite actualizar las probabilidades de sus posibles ubicaciones. Entre los métodos probabilísticos más usados hoy en día cabe citar el Filtro de Kalman, el Filtro de Kalman Extendido, el Método de Monte Carlo, la Localización de Markov,etc.

A continuación explicaremos algunos de los métodos de localización a los que hemos hecho alusión en la primera clasificación.

Localización odométrica

Se denomina odometría al estudio de la estimación de la posición de vehículos con ruedas durante la navegación. Para realizar esta estimación se usa información sobre la rotación de las ruedas a lo largo del tiempo. Este término también suele usarse para referirse a la distancia que se ha recorrido. La palabra en sí, se compone de la palabras griegas *hodos* (trayecto) y *metron* (medida).

Los robots móviles usan la odometría para estimar (y no determinar) su posición relativa a su localización inicial. La odometría proporciona una buena precisión a corto plazo, es barata de implementar y permite tasas de muestreo muy altas. Sin embargo, la idea fundamental de la odometría es la integración de información incremental del movimiento a lo largo del tiempo lo cual conlleva una inevitable acumulación de errores. En concreto, la acumulación de errores de orientación además de grandes errores en la estimación de la posición los cuales van aumentando proporcionalmente con la distancia recorrida por el robot.

En la figura 2.8, observamos como la desviación que se obtiene si sólo tenemos información de la odometría pura es mayor que si obtenemos información a través de un escáner láser e intentamos correguir mediante estimación la trayectoria.

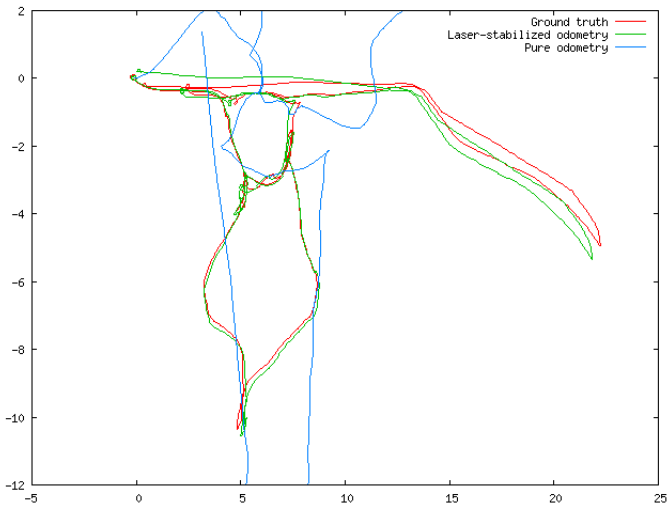


Figura 2.8: *Representación de la trayectoria: odometría frente a estimación odométrica mediante láser.*

A pesar de estas limitaciones, muchos investigadores están de acuerdo en que la odometría es una parte importante del sistema de navegación y que debe usarse con medidas del posicionamiento absolutas para proporcionar una estimación de la posición más fiable.

La odometría se basa en ecuaciones simples que se pueden implementar fácilmente y que utilizan datos de encoders situados en las ruedas del robot. Sin embargo, la odometría también está basada en la suposición de que las revoluciones de las ruedas pueden ser traducidas en un desplazamiento lineal relativo al suelo, suposición que no tiene una validez absoluta. Un ejemplo extremo es cuando las ruedas patinan: si por ejemplo, una rueda patina sobre una mancha de aceite y la otra no, entonces se registrarán las revoluciones en la rueda a pesar de que éstas no correspondan a un desplazamiento lineal de la misma.

Los errores se pueden agrupar en dos categorias : errores sistemáticos y errores no sistemáticos.
Errores sistemáticos. Entre los que cabe destacar:

- Incertidumbre en el diámetro de las ruedas.
- Mal alineamiento de las ruedas.
- Incertidumbre en la distancia entre los puntos de apoyo de las ruedas.

Errores no sistemáticos. Se clasifican:

1. Desplazamiento en suelos desnivelados.
2. Desplazamiento sobre objetos inesperados que se encuentran en el suelo.
3. Patinaje de las ruedas debido a:
    - Suelos resbaladizos.
    - Sobre-aceleración.
    - Derrapes (debidos a una rotación excesivamente rápida).
    - Fuerzas externas (interaciión con cuerpos externos).

Una clara distinción entre errores sistemáticos y no sistemáticos es de gran importancia a la hora de reducir los errores en odometría. Por ejemplo, los errores sistemáticos se pueden corregir con una buena calibración, aunque en superficies no rugosas de entornos interiores son éstos los que contribuyen más a los errores de odometría que los no sistemáticos. En cambio, en supericies abruptas con irregularidades significatvas, son los no sistemáticos los que predominan.

El principal problema de los errores no sistemáticos es que pueden aparecer inesperadamente (por ejemplo, cuando el robot pasa por encima de un objeto que se encuentra en el suelo como puede ser un cable), y pueden causar errores muy grandes en la estimación de la posición.

En la determinación de la incertidumbre que se produce a la hora de estimar la posición de un robot que utiliza odometría, cada posición calculada está rodeada por una *elipse de error* característica, la cual indica la región de incertidumbre para la posición actual del robot, como se puede apreciar en la figura 2.9.

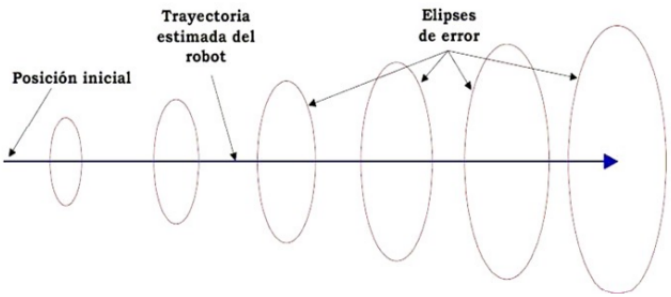


Figura 2.9: *Elipses de error en torno a la trayectoria estimada de un robot.*

Estas elipses crecen a medida que la distancia recorrida aumenta, a no ser que un sistema de estimación absoluto ponga a cero su tamaño. Estas técnicas de estimación del error se basan en parámetros derivados de los errores sistemáticos puesto que las magnitudes de los no sistemáticos son impredecibles.

Localización mediante balizas

Estos sistemas permiten determinar la posición del robot mediante emplazamientos de un número de balizas de posición conocida en el entorno (*landmarks*). Aunque puede entenderse que este proceso conlleva la percepción del entorno, la posición no se estima a partir del análisis o interpretación del mismo, sino que se determina en base al principio de triangulación, bien a partir de las medidas de distancias, de ángulos ó combinaciones de las dos. El número mínimo de balizas requeridas dependerá del tipo de sistema empleado.

Para el caso de navegación en el plano (2D), los ángulos de observación con respecto al eje x de dos balizas son suficientes (Figura 2.10a). Sólo serán necesarias tres de ellas si las medidas se refieren a ángulos entre ellas (Figura 2.10b). Cuando la información que se maneja son distancias, la solución obtenida utilizando dos balizas puede ser insuficiente si no se dispone de alguna información adicional que permita resolver la ambigüedad resultante (Figura 2.10c). Por último, también pueden combinarse información angular y de distancia, siendo en este caso necesaria la observación de una única baliza (Figura 2.10d), aunque para ello se requiere un sensor más complejo que sea capaz de obtener ambas medidas.

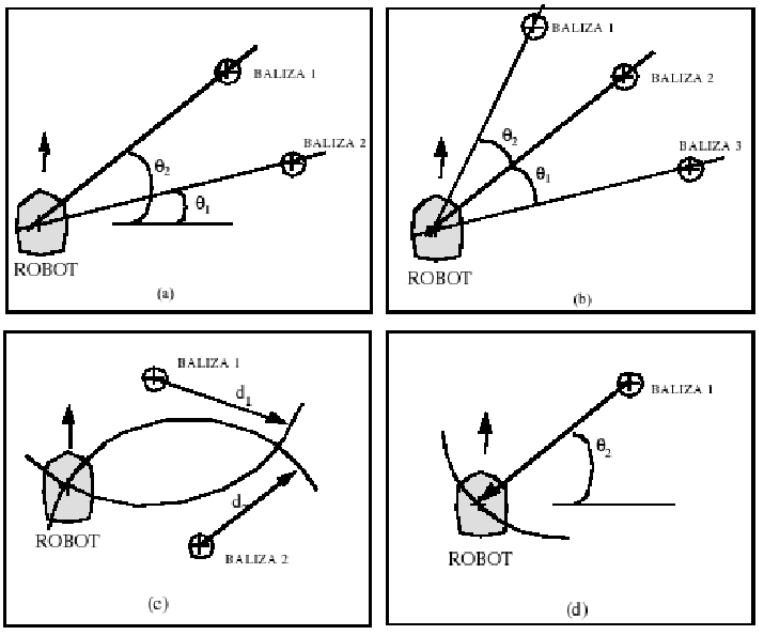


Figura 2.10: *Diferentes modelos de triangulación: a)Ángulos absolutos de referencia de las balizas.b)Ángulos observados entre balizas.c)Distancias a las balizas.d)Ángulos y distancia a una baliza.*

Las técnicas descritas anteriormente corresponden a situaciones ideales, en donde las medidas y las posiciones de las balizas están libres de errores. En la práctica, para minimizar el error cometido en la estimación se usan un mayor número de ellas.

Existen diversas maneras de implantar este tipo de localización en los robots móviles. Una posibilidad es instalar en el vehículo un receptor giratorio que rastree el entorno en busca de señales emitidas por las balizas, un ejemplo de este tipo consiste en una serie de balizas de luz infrarroja y un dispositivo óptico giratorio en el robot capaz de detectar este tipo de luz. Conocida la velocidad de giro del receptor óptico, el sistema determina los ángulos entre balizas consecutivas a partir del tiempo que transcurre entre las detecciones de ésta. La posición del robot se estima a partir de estos ángulos mediante relaciones trigonométricas, mientras que la orientación se obtiene directamente midiendo el ángulo entre cualquiera de las balizas del entorno y otra colocada abordo con tal fin.

Una configuración alternativa consiste en situar en el entorno un conjunto de marcas especiales, señales luminosas, códigos de barras , etc, y dotar al vehículo de una o varias cámaras CCD que exploren el entorno en busca de ellas. La precisión y fiabilidad de este tipo de estimación depende fundamentalmente del tipo de señal (infrarrojos, láser, radio, ultrasonidos, etc), de las caracteríticas del sensor y del número de balizas utilizadas en la triangulación. En general, este método está considerado como uno de los más precisos.

Las principales desventajas radican en la necesidad tanto de configurar apropiadamente el entorno de trabajo, como de garantizar que un suficiente número de estas señales queden en todo momento libres de oclusiones y dentro del campo visual del sensor. Debe tenerse en cuenta los problemas que originan las condiciones ambientales de iluminación y ruido (acústico, electromagnético,etc). Todo ello impide la utilización de esta técnica en entornos muy dinámicos o no estructutados.

Sistema de posicionamiento global

Cuando la localización debe realizarse en exteriores, el GPS (Global Position System) es la mejor opción para localizar el sistema ya que puede hayar un objeto móvil en cualquier punto de la superficie terrestre con gran precisión.

Figura 2.11: *Sistema de posicionamiento global basado en 24 satélites.*

La aparición de este sistema data de 1978 cuando se puso en órbita el primer satélite de la serie Navstar. El GPS funciona mediante una red de 24 satélites (21 operativos y 3 de respaldo) en órbita sobre el globo a 20.200 km con trayectorias sincronizadas para cubrir toda la superficie de la tierra (Figura 2.11). Colocados a bordo de los satélites viajan relojes atómicos, lo que permite al sistema GPS ser muy preciso en la escala de tiempo.

Para que el receptor GPS determine su posición, localiza automáticamente cuatro satélites de la red de los cuales recibe señales indicando la posición y reloj de éstos (Figura 2.12). El receptor GPS calcula las distancias a los satélites a partir de los retardos de las señales recibidas y por triangulación calcula la altitud, latitud y altura a la que estos se encuentran. Conocidas las distancias a los satélites y las posiciones relativas entre ellos, se determina fácilmente la posición del receptor.

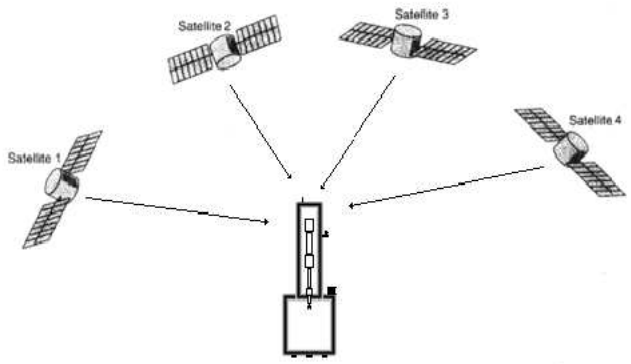


Figura 2.12: *Localización por GPS.*

Las fuentes de error en el GPS no son acumulativas, como ocurre en el caso de los encoders en la localización odométrica. Estos errores pueden ser relativos a la órbita del satélite (máximo 1 metro), errores de modelización de la propagación en la troposfera y la ionosfera (1 metro y 10 metros respectivamente), errores debidos a rebotes de la señal (0.5 metros), errores de usuarios o de configuración en el receptor (los más habituales, pudiendo ir desde 1metro a cientos de kilometros).

Sistema de navegación inercial

Los sistemas de navegación inercial (INS) estiman la posición y orientación del robot empleando medidas de las aceleraciones y ángulos de orientación. La primera integración de las aceleraciones proporciona la velocidad y la segunda la posición. Los acelerómetros suelen estar basados en sistemas pendulares, su precisión resulta crítica ya que incluso pequeños errores cometidos por éste repercuten notablemente en la posición estimada debido a la doble integración de las aceleraciones. En numerosos robots móviles las aceleraciones son pequeñas con lo que la relación señal/ruido es también pequeña lo que complica la estimación.

Para medir los ángulos de orientación se emplean giróscopos que pueden ser mecánicos u ópticos (de anillo de láser o de fibra óptica), siendo también posible medir el ángulo de orientación empleando brújulas. A diferencia de los sistemas odométricos, los sistemas de navegación inerial no se ven afectados por los problemas derivados de la interacción del vehículo con el suelo, además pueden corregir los efectos de las ondulaciones e irregularidades del terreno. Es-

to hace que en la práctica sean mucho más fiables y precisos que los basados en la odometría, aunque como contrapartida hay que decir que son más frágiles y caros.

Sonar

Uno de los sensores de mayor uso para la percepción del entorno es el sonar. El sonar, es un sensor basado en el tiempo de vuelo de una señal de ultrasonido que es transmitida continuamente en forma de pulso, la cual es captada de nuevo al chocar con algún objeto y rebotar. Mediante física elemental, la distancia al objeto es determinada multiplicando la velocidad de la onda por el tiempo que tarda en recorrer la distancia [2].

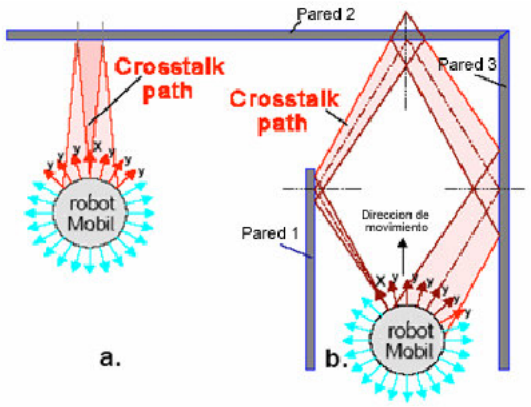


Figura 2.13: *Interferencia entre sonares(Crosstalk). Puede ser directo(a) o indirecto(b).*

El uso del sonar tiene la ventaja de un reducido costo y una elevada velocidad de respuesta. Pero tiene algunas desventajas como la de obtener información muy reducida, ya que solamente adquiere la distancia sin mayor información sobre el obstáculo como puede ser color o textura. Además, se producen rebotes de la señal obteniendo falsas lecturas por lo que un robot utiliza varios sonares dispuestos en forma de anillos, como por ejemplo en [14], método que reduce la posibilidad de errores por rebotes falsos pero que introduce posibles interferencias con otros sonares (Crosstalk) como se aprecia en la figura 2.13.

Adquisición 3D

A partir de la información que se obtiene con un sonar sólo se pueden construir mapas en dos dimensiones. Debido a que el medio con el que interactúa el robot es tridimensional surge la necesidad de utilizar sensores más sofisticados que obtengan mayor información del entorno. En este sentido, se puede hacer uso de las tecnologías de adquisición 3D para obtener un ambiente tridimensional del entorno del robot. Las dos tendencias de mayor uso con respecto a la adquisición 3D son: *escáner láser y visión estéreo* [3].

1. Escáner láser.

Los métodos de escáner láser se dividen en dos grupos: indirectos y directos. Los *indirectos* consisten en utilizar un patrón de luz láser y proyectarlo sobre una superficie determinada mientras es adquirida por medio de una cámara. Las coordenadas tridimensionales de un punto determinado pueden obtenerse de acuerdo a las diferencias en los patrones de luz reflejados en la superficie. Los *métodos directos* consisten en emitir un pulso de luz por un láser que es reflejado al incidir sobre una superficie, en este caso la distancia al punto deseado se calcula determinando el retardo entre la emisión y la recepción de la señal.

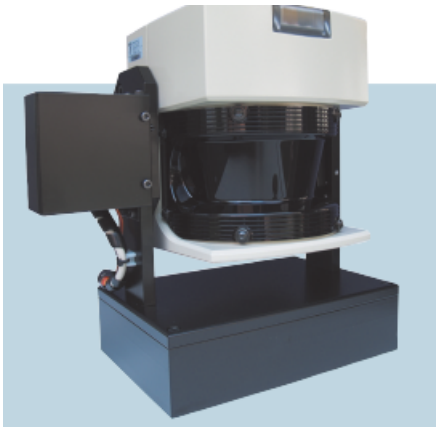


Figura 2.14: *Escáner láser 3D*

El uso de los métodos de escáner láser permite obtener información tridimensional con una alta precisión, pero presenta una serie de inconvenientes , como un alto costo de equipos, alto tiempo de procesamiento, errores ante superficies reflectantes y la imposibilidad de obtener información sobre el color.

2. Adquisición 3D mediante visión.

A partir de una imagen bidimensional no se puede obtener una escena 3D sin ninguna información extra, para lograrlo se puede elegir entre dos opciones: conocer características del entorno ó tomar más imágenes (visión estéreo).

La primera de ellas no la trataremos pues no es usada en robótica. En cuanto a la visón estéreo, se refiere a la obtención de dos o más imágenes desde diferentes vistas de un objeto de interés con el objetivo de encontrar sus coordenadas XYZ. Ésto se obtiene en base a la configuración en el espacio físico (ubicación y orientación) de las cámaras con las que se obtienen las imágenes y a las diferencias existentes entre las imágenes adquiridas.

La adquisición mediante visión 3D permite obtener información tridimensional con una precisión bastante aceptable y con una alta velocidad de respuesta. La principal desventaja radica en la sensibilidad a los cambios de iluminación.

Capítulo 3

Localización y Modelado Simultáneo

En este capítulo se presentan las bases matemáticas del problema de Localización y Modelado Simultáneo de robots móviles, se desarrollan las expresiones que modelan el problema del SLAM así como algunos de los algoritmos que han sido diseñados para solucionar este problema.

SLAM

El problema de SLAM empezó a ser tratado en profundidad cuando los problemas de mapeo y navegacion empezaron a ser muy complicados y en ciertos casos imposibles de resolver si se hacían por separado.

Los trabajos de Smith y Cheeseman [31] y Durrant-Whyte [9] establecieron las bases estadísticas para describir relaciones espaciales entre los diferentes elementos del entorno y manipular las incertidumbres geométricas asociadas a sus respectivas posiciones. Poco después, el artículo de Smith, Self y Cheeseman [30]mostró cómo a medida que un robot móvil se desplaza por un entorno desconocido, realizando observaciones de objetos relativas a su propio sistema de referencia, las estimaciones de las posiciones de dichos objetos quedan necesariamente correladas. Esto se debe a la parte del error común a todas las medidas que tiene su origen en la propia incertidumbre de la estimación de la posición.

Las soluciones que mejores resultados han obtenido a la hora de abordar el problema del SLAM son aquellas basadas en técnicas probabilísticas, que consiguen hacer frente a todas las fuentes de incertidumbre involucradas en el proceso. Este tipo de algoritmos tienen su base en el torema de Bayes, que relaciona entre sí las probabilidades marginal y condicional de dos variables aleatorias.

3.1. La formulación Bayesiana del SLAM

El principal obstáculo a la hora de modelar un entorno desconocido utilizando un robot móvil para su exploración es, la inevitable incertidumbre que se deriva de la imperfección de los sensores y modelos empleados. Si los sensores fueran perfectos, proporcionarían medidas absolutamente precisas de los objetos detectados, que podrían ser insertados en un mapa en su posición exacta respecto a un sistema de referencia ligado al robot.

Del mismo modo, al desplazarse la máquina, una medida exacta del giro de sus ruedas combinada con un modelo exacto de su movimiento, nos permitirían determinar exactamente la posición del robot cuando este realizase una nueva medida. Podríamos así construir incrementalmente un modelo perfecto del entorno.

Desafortunadamente tanta perfección no existe en este mundo; o al menos no en el mundo que nos ofrece nuestro actual desarrollo tecnológico. Se impone por tanto la necesidad de acomodar estas incertidumbres en las soluciones obtenidas. La manera evidente de hacerlo es considerar que tanto la posición del robot como los elementos que modelan su entorno son variables aleatorias. Así, los algoritmos existentes modelan ambos de manera probabilística, y utilizan métodos de inferencia para determinar aquella configuración que es más probable teniendo en cuenta las medidas que se van obteniendo.

El principio básico que subyace en cualquier solución exitosa del SLAM es la regla de Bayes. Para dos variables aleatorias,x y d, esta regla establece de manera muy compacta lo siguiente:

$$p(x \mid d) = \frac{p\,(d \mid x)\,p(x)}{p(d)} \tag{3.1}$$

La anterior ecuación constituye un mecanismo básico de inferencia. Supongamos que queremos obtener información acerca de la variable x -por ejemplo, al estado de un sistema compuesto por un mapa y un robot- basándonos en la información contenida en otra variable d -que podría ser un conjunto de medidas adquiridas por un sensor-. La anterior regla nos indica que este problema se puede resolver simplemente multiplicando dos términos.

- El modelo generativo $p(x \mid d)$, que expresa la probabilidad de obtener la medida d bajo la hipótesis expresada por el estado x, y
- el grado de confianza que damos a que x sea precisamente el caso antes de recibir los datos $p(x)$.

Es importante el hecho de que el denominador de la ecuación 3.1 no depende de la variable que pretendemos estimar, x. Por esta razón $p(d)^{-1}$ se suele escribir como un factor de normalización en la regla de Bayes, generalmente expresado mediante la letra η. Obtenemos así una expresión algo más compacta para la regla anterior:

$$p\left(x \mid d\right) = \eta p\left(d \mid x\right) p(x) \tag{3.2}$$

El problema del mapeado de un entorno está sujeto a una variable adicional no contemplada por la formulación clásica del teorema de Bayes, el tiempo. Durante la labor exploratoria de un robot móvil, los datos se adquieren secuencialmente en el tiempo. Tanto las medidas propioceptivas, a las que denominaremos con la letra u- las cuales miden el desplazamiento del robot-, como las medidas exteroceptivas obtenidas por los sensores disponibles a bordo de la máquina, z, son adquiridas a lo largo de todo el proceso.
Sin pérdida de generalidad podemos asumir que esta información se recibe alternativamente según la secuencia:

$$z_1,\ u_1,\ z_2,\ u_2,\ ...,\ z_t,\ u_t,\ ...$$

donde los subíndices expresan un instante temporal concreto.
La variable u puede representar desde la señal de control enviada a los actuadores de la máquina, hasta las medidas proporcionadas por los encóders acoplados a las ruedas del robot o a las suministradas por sensores de navegación inercial (IMUs) como acelerómetros o giróscopos.

En el primer caso, será el modelo de la máquina el que determine el desplazamiento realizado, mientras que en el segundo y tercer casos será la propia medida, o su integración a lo largo del tiempo, la que proporcione esta estimación.

La generalización temporal del teorema de Bayes, en la cual se fundamentan todas las soluciones probabilísticas del SLAM, reciben el nombre de *Filtro de Bayes*. Se trata de un estimador recursivo que calcula la secuencia de distribuciones de probabilidad *a posteriori* (tras recibir los datos) de magnitudes que no pueden ser directamente observadas, en función de otras que sí lo son. Así, la formulación genérica del filtro de Bayes calcula la probabilidad del estado en un instante dado x_t utilizando la siguiente ecuación recursiva:

$$p(x_t \mid z^t,\, u^t) = \eta p\,(z_t \mid x_t) \int p(x_t \mid u_t,\, x_{t-1}) p(x_{t-1} \mid z^{t-1},\, u^{t-1}) dx_{t-1} \tag{3.3}$$

En este caso se utiliza el superíndice de *t* para indicar el conjunto de todas las medidas acumuladas hata ese preciso instante:

$$z^t = \{z_1,\, z_2,\, ...,\, z_t\} \tag{3.4}$$

$$u^t = \{u_1,\, u_2,\, ...,\, u_t\} \tag{3.5}$$

La anterior ecuación 3.3 expresa una relación recursiva, puesto que la probabilidad $p(x_t \mid z^t,\, u^t)$ se calcula en cada momento haciendo uso de la misma probabilidad obtenida un instante termporal anterior, $p(x_{t-1} \mid z^{t-1},\, u^{t-1})$. De esta manera se evita la necesidad de mantener almacenadas todas las medidas $\{z_i,\, i = 1,\, ...,\, t\}$ y $\{u_i,\, i = 1,\, ...,\, t\}$ a lo largo del proceso que, según la anterior expresión, se puede mantener indefinidamente en el tiempo.

En el problema de la localización y modelado simultáneo, el estado del sistema x_t está compuesto en cada instante por la posición del robot, a la que denominaremos s_t, y por las posiciones del conjunto de objetos incluidos en el mapa,m_t.

$$x_t \equiv \{s_t,\, m_t\} \tag{3.6}$$

Por lo tanto el filtro de Bayes adaptado al problema que nos ocupa se puede escribir como:

$$p(s_t,\, m_t \mid z^t,\, u^t) = \eta p\,(z_t \mid s_t,\, m_t) \int p(s_t,\, m_t \mid u_t,\, s_{t-1},\, m_{t-1}) p(s_{t-1},\, m_{t-1} \mid z^{t-1},\, u^{t-1})\, ds_{t-1}\, dm_{t-1} \tag{3.7}$$

Asumiendo que el mundo es estático -o, al menos, la parte de él que se pretende modelar- podemos omitir el subíndice temporal al referirnos al mapa, ($m_t \equiv m$).

Esta expresión, para servir de utilidad práctica, requiere de dos elementos fundamentales. Se trata de dos modelos generativos que dependen del propio robot, y de la relación perceptual que este establece con su entorno:

- El **modelo de percepción** $p(z_t \mid s_t,\, m)$, que describe en términos probabilístcos cómo se generan las medidas z_t en función de la posición del robot y la configuración del mapa.
- El **modelo de movimiento** $p(s_t \mid u_t,\, s_{t-1})$, que determina la probabilidad de que el robot llegue a la posición s_t desde otra s_{t-1}, cuando se aplica la acción u_t (o, equivalentemente, cuando se mida un desplazamiento dado por esta variable).

Ahora bien, el mecanismo de inferencia propuesto por la ecuacion 3.7, si bien es cierto que se adapta a nuestras necesidades desde el punto de vista lógico, presenta el grave inconveniente de no admitir una solución cerrada, o que sea fácilmente calculable utilizando un computador. Así pues, es preciso realizar simplificaciones o suposiciones que faciliten su uso práctico.

La manera de establecer estas hipótesis adicionales a la hora de implementar el filtro de Bayes es el rasgo que diferencia a los múltiples algoritmos de construcción de mapas, que a continuación se describen brevemente.

3.1.1. Principales algoritmos.

3.1.1.1. Filtro de Kalman.

Uno de los casos típicos de redes Bayesianas se da cuando el estado oculto (no medible) es modelado con Gaussianas. En 1960 R.E.Kalman publicó un método con una solución recursiva, el cual tuvo gran éxito con la aparición de la computación [40]. Los filtros de Kalman, como se llama a dicha técnica, son la herramienta con más fundamento matemático y con resultados más exactos a la hora de hacer un seguimiento del estado en una red Bayesiana dinámica, lo que los convierten en la técnica a utilizar en este tipo de problemas.

Viendo el algoritmo a "alto nivel", los filtros de Kalman estiman un proceso usando una forma de control de retroalimentación: el filtro estima el estado del proceso de un instante dado y luego obtiene retroalimentación en forma de mediciones ruidosas. Las ecuaciones para el filtro de Kalman se clasifican en dos grupos: a) ecuaciones de *actualización de estado* y b) ecuaciones de *actualización de medición*.

– Las ecuaciones de acualización de estado son responsables de proyectar al futuro la estimación del estado actual y la estimación de la covarianza del error para obtener el estimado a *priori* del siguiente instante.
–Las ecuaciones de actualización de medición son responsables de la retroalimentación, al incorporar a la estimación a *priori* una nueva medición para obtener un estado a *posteriori* mejorado. Las ecuaciones de actualización pueden ser visualizadas como ecuaciones *predictoras*, mientras que las actualizaciones de medición pueden ser vistas como ecuaciones *correctoras*.

Una crítica a los filtros de Kalman es que sólo pueden estimar el estado de un proceso controlado en tiempos discretos dirigidos por una ecuación lineal estocástica. Entonces ¿qué ocurre si el proceso a estimar y/o la relación de medición con el proceso es no-lineal? Un filtro de Kalman que linealiza sobre la actual media y covarianza se conoce como *filtro extendido de Kalman o EKF* por sus siglas en inglés. Es análogo a una serie de Taylor, se linealiza la estimación alrededor del actual valor estimado usando las derivadas parciales de las funciones del proceso y la medición para calcular estimados aún frente a relaciones no-lineales.

Es importante destacar que un punto débil de los EKF es que las distribuciones (o densidades en el caso contínuo) de varias variables aleatorias no son normales después de llevar a cabo su respectiva transformación no lineal. Los EKF son simplemente una estimación *ad hoc* del estado que sólo aproxima a la optimalidad de la regla de Bayes por linealización.

Filtros de Kalman aplicados al SLAM

El enfoque más común para resolver el problema de Mapeado y Localización Simultánea es utilizar filtros Kalman [36]. Es más, la literatura designa a los algoritmos SLAM como aquellos que están basados en filtros Kalman, aunque SLAM es el nombre de un problema y no de una solución.

La principal ventaja del enfoque del filtro Kalman es el hecho de que estima la distribución posterior completa de los mapas en línea. Además mantiene toda la incertidumbre en el mapa $O(K^2)$lo cual puede ser beneficioso para la navegación. Adicionalmente, el enfoque puede demostrar que converge con probabilidad de uno al verdadero mapa y posición del robot sobre una distribución de incertidumbre residual que se origina en gran medida de una fluctuación original aleatoria.

A pesar de todas estas ventajas, la solución al problema de mapeado utilizando EKF sufre tres importantes limitaciones[33]:

1. La complejidad para cada actualización es de $O(K^2)$, donde K es el número de características que describen el mapa, aún con la ausencia del problema de asociación de datos. Esta limitación impone una importante restricción al escalar.

2. Los EKF no pueden incorporar información negativa, es decir, no pueden usar el hecho de que un robot pueda no ver una característica aunque esta sea esperada. La razón de esta incapacidad es que las mediciones negativas producen distribuciones posteriores no Gaussianas, que no pueden representarse en los EKF.

3. Los EKF no proveen soluciones al problema de la asociación de datos. La falsa asociación de datos conduce frecuentemente a errores catastróficos en el mapeado.

Entre las desventajas también se encuentra la costosa operación en la actualización del filtro de Kalman a la hora de multiplicar matrices, las cuales pueden ser implementadas en $O(K^2)$, donde K es el número de características en el mapa [36]. En la práctica el número de características no se conoce a priori, por lo que si el tamaño del ambiente a modelar es grande, a medida que la ruta del robot es mayor, la actualización se va complicando cada vez más.

Otra de las limitaciones más importantes del filtro de Kalman es la suposición del ruido Gaussiano. En particular, la suposición de que el ruido de medición debe ser independiente y Gaussiano impone una limitación clave, que tiene importantes implicaciones en el momento de implementarlo. Por ejemplo, en un ambiente con dos hitos exactamente iguales; medir tales hitos conducirá a una distribución multimodal sobre las posibles posiciones del robot.

De manera más general podemos decir que los filtros de Kalman son incapaces de hacer frente al problema de la asociación de correspondencias, que consiste en asociar mediciones individuales de un sensor con características en el mapa.

3.1.1.2. Método de Monte Carlo

En el caso de las técnicas de muestreo, como pueden ser los algoritmos de aproximación, se basan en asumir aleatoriamente la existencia de valores para algunos nodos y luego usar estos valores para inferir en la cantidad de los otros nodos. Luego, se mantienen estadísticas de los valores que estos nodos toman, y finalmente estas estadísticas dan la respuesta [6]. A estos métodos se les llama técnicas de Monte Carlo.

El uso de los métodos de Monte Carlo como herramienta de investigación proviene del trabajo realizado en el desarrollo de la bomba atómica durante la segunda guerra mundial en el Laboratorio Nacional de Los Álamos en EE.UU. Fue en 1930 cuando Enrico Fermi y Stanislaw Ulam desarrollaron las ideas básicas del método, más tarde, a principios de 1947 John von Neumann envió una carta a Richtmyer a Los Álamos en la que expuso de modo exhaustivo tal vez el primer informe por escrito del método de Monte Carlo.Una de las primeras aplicaciones de este método a un problema determinista fue llevada a cabo en 1948 por Enrico Fermi, Ulam y von Neumann cuando consideraron los valores singulares de la ecuación de Schrödinger.

El problema fundamental al que se enfrentan es encontrar el valor esperado de alguna función $f(x)$ con respecto a una distribución de probabilidad $p(x)$. Aquí, x puede estar formada por variables discretas, contínuas, o alguna combinación de ambas. En el caso de variables continuas se quiere calcular el valor esperado en la ecuación 3.8.

$$\langle f \rangle = \int f(x)p(x)dx \tag{3.8}$$

Esto se ilustra esquemáticamente para una distribución de una variable en la (Figura 3.1).

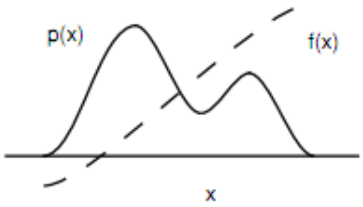


Figura 3.1: *Función $f(x)$ cuyo valor esperado será evaluado con respecto a una distribución $p(x)$.*

La idea principal de este algoritmo se basa en representar la función de densidad de probabilidad de la posición del robot mediante un conjunto finito de partículas, $\{x^{(i)}, P(x^{(i)})\}$, $i = \{1, ..., N\}$, donde x^i representa la ubicación (x, y, Θ) del robot y $P(x^i)$ su probabilidad asociada, calculada mediante un modelo de observación. Se asume que $\sum_{i=1}^{N} P(x^i) = 1$.

Inicialmente, el conjunto de muestras se distribuye uniformemente de forma aleatoria por todo el espacio de posiciones posibles.Las muestras se generan en cada instante,
$M_t = \{x_t^{(1)}, x_t^{(2)}, ..., x_t^{(n)}\}$, a partir de las muestras del instante anterior,
$M_t = \{x_{(t-1)}^{(1)}, x_{(t-1)}^{(2)}, ..., x_{(t-1)}^{(n)}\}$, junto con las mediciones del entorno z_t, y las acciones ejecutadas por el robot, $u_{(t-1)}$ (desplzamiento en x,y,Θ) siguiendo tres pasos recursivamente:

Predicción

Se desplazan las muestras según la acción ejecutada y se estima la nueva posición de las muestras. Sea u_{t-1} la acción ejecutada por el robot en el instante $t-1$. Para cada muestra $x_{t-1}^{(i)} \in M_{t-1}$ se predice su nuevo estado en el instante t, añadiendo un cierto error de movimiento:

$$x_t^{(i*)} = x_{t-1}^{(i)} + u_{t-1} + (N(0,\, \sigma_x),\, N(0,\, \sigma_y),\, N(0,\, \sigma_\Theta))^T \tag{3.9}$$

con $i = \{1,\, ...,\, N\}$. De esta forma construimos un nuevo conjunto de N muestras, M_t^*, para el instante t.

Actualización de observaciones

Se calculan las probabilidades a posteriori de cada muestra, dada la última observación sensorial. Sea z_t la observación realizada por el robot en el instante actual t. Para cada muestra $x_t^{(i*)} \in M_t^*$ se actaliza su probabilidad asociada, $P(x_t^{(i*)})$, según la verosimilitud de que la observación z_t haya sido obsevada desde el estado $x_t^{(i*)}$.

$$P(x_t^{(i*)}) = P(x_t^{(i*)} \mid z_t),\ i = \{1,\, ...,\, N\} \tag{3.10}$$

Remuestreo

Se genera un nuevo conjunto de muestras con distribución proporcional a la verosimilitud de las muestras anteriores. Construir un nuevo conjunto de N muestras, M_t, remuestreando con sustitución el conjunto M_t^*, de forma que escoja cada muestra $x_t^{(i*)}$ con probabilidad proporcional a la verosimilitud de la misma $P(x_t^{(i*)})$.

$$(x_t^{(i)},\, P(x_t^{(i)})) \longleftarrow \textit{Escoger muetras de } M_t^* \tag{3.11}$$

con $i = \{1,\, ...,\, N\}$. Normalizar las probabilidades $P(x_t^{(i)})$ de las muestras de M_t de forma que $\sum_{i=1}^{N} P(x^i) = 1$. Para ello:

$$P(x_t^{(i)}) \longleftarrow \frac{P(x_t^{(i)})}{\sum_{j=1}^{N} P(x^i)},\ i = \{1,\, ...,\, N\} \tag{3.12}$$

Siguiendo esta forma de proceder, las muestras se van concentrando alrededor de las posiciones más probables del robot, hasta que finalmente se produce la convergencia de la población de muestras en un área reducida alrededor de la posición real del sistema móvil.

3.1.1.3. Filtro de partículas

El algoritmo de filtro de partículas es un método Monte Carlo que conforma la base para la mayoría de los filtros Monte Carlo desarrollados. A los filtros de partículas también se les conoce como secuencias Monte Carlo, muestreo secuencial con importancia (SIS por sus siglas en inglés -*Sequencial Importance Sampling*-), *bootstrap filter*, algoritmo de condensación, supervivencia por adapatación,etc.

La idea fundamental es representar la función de distribución posterior buscada con un conjunto de muestras aleatorias con pesos asociados y calcular estimados basados en estas muestras y pesos [17]. Al aumentar el número de muestras, esta caracterización Monte Carlo se vuelve una representación equivalente de la descripción funcional usual de la función de distribución posterior, y el filtro SIS se acerca al estimador Bayesiano óptimo.

Para detallar estas ideas se denota $\{x^i_{0:t},\, w^i_t\}^{N_s}_{i=1}$ como *mediciones aleatorias* que caracterizan la función de distribución posterior $p(x_{0:t} \mid z_{1:t})$, donde $x^i_{0:t}$, $i = \{1, ..., N_s\}$ es un conjunto de puntos de soporte con pesos asociados x^i_t, $i = \{1, ..., N_s\}$ y $x_{0:t} = \{x_j,\, j = 0, ..., t\}$ es el conjunto de todos los estados hasta el tiempo t. Los pesos están normalizados de tal forma que $\sum_i\ w^i_t = 1$. La densidad posterior en t puede ser aproximada como en la ecuación 3.13

$$p(x_{0:t} \mid z_{1:t}) \approx \sum_{i=1}^{N_s}\ w^i_t\, \delta\left(x_{0:t} - x^i_{0:t}\right) \tag{3.13}$$

donde la función δ es la función de Dirac.

En otras palabras, se selecciona aleatoriamente un conjunto de muestras $x^i_{0:t}$ obtenidas del superconjunto $x_{0:t}$ que representan todos los estados posibles en una distribución fácilmente muestreable. Las muestras seleccionadas son aceptadas o no como parte de la distribución posterior buscada (razón del uso de la función de Dirac) en base a la ponderación de la muestra.

Este es un enfoque no paramétrico, y por tanto puede manejar distribuciones no lineales, multimodales,etc. La ventaja sobre la discretización es que el método es adaptativo, colocando más partículas (correspondiendo a una discretización más fina) en lugares donde la densidad probabilística es mayor.

Los pesos son elegidos utilizando el principio de *Importancia del Muestreo* [17]. Este principio se fundamenta en lo siguiente: suponiendo $p(x) \propto \pi(x)$, donde $p(x)$ es una densidad de probabilidad difícil de muestrear, pero para la cual $\pi(x)$ puede ser fácilmente evaluada (y por tanto $p(x)$ por proporcionalidad). También sea $x^i \sim q(x)$, $i = \{1, ..., N_s\}$ muestras que son fácilmente generadas por una distribución propuesta $q(x)$, llamada una *densidad de importancia*. Entonces, una aproximación ponderada a la densidad $p(x)$ está dada por:

$$p(x) \approx \sum_{i=1}^{N_s} w^i \delta(x - x^i) \tag{3.14}$$

donde

$$w^i \propto \frac{\pi(x^i)}{q(x^i)} \tag{3.15}$$

es el peso normalizado de la i-ésima partícula.

Por lo tanto, si las muestras $x^i_{0:t}$, son esbozados a partir de una densidad de importancia, $q(x_{0:t} \mid z_{1:t})$, entonces los peso en 3.13 definidos en 3.15 serán:

$$w^i_t \propto \frac{p(x^i_{0:t} \mid z_{i:t})}{q(x^i_{0:t} \mid z_{1:t})} \tag{3.16}$$

Para poder calcular la densidad de manera secuencial y manejarla como un filtro con propiedad Markoviana, para cada iteración, se deberían tener muestras que constituyen una aproximación $p(x_{0:t-1} \mid z_{i:t-1})$, y desear una aproximación $p(x^i_{0:t} \mid z_{i:t})$ con un nuevo conjunto de muestras. Si la densidad de importancia es seleccionada para factorizar tal casualidad de manera que:

$$q(x_{0:t} \mid z_{1:t}) = q(x_t \mid x_{0:t-1},\, z_{1:t})\; q(x_{0:t-1} \mid z_{1:t-1}) \tag{3.17}$$

entonces se pueden obtenre muestras $x^i_{0:t} \sim q(x_{0:t} \mid z_{1:t})$ aumentando cada una de las muestras existentes, $x^i_{0:t-1} \approx q(x_{0:t-1} \mid z_{1:t-1})$, con el nuevo estado $x^i_t \approx\ q(x_t \mid z_{1:t})$.

Para poder deducir la ecuación de actualización de pesos, la distribución $p(x_{0:t} \mid z_{1:t})$ debe expresarse pimero en términos de $p(x_{0:t-1} \mid z_{1:t-1})$, $p(z_t \mid x_t)$ y $p(x_t \mid x_{t-1})$:

$$p(x_{0:t} \mid z_{1:t}) =$$

$$= \frac{p(z_t|x_{0:t}, z_{1:t-1})\, p(x_{0:t-1}|z_{1:t-1})}{p(z_t|z_{1:t-1})}$$

$$= \frac{p(z_t|x_{0:t}, z_{1:t-1})\; p(x_t|x_{0:t}, z_{1:t-1})\; p(x_{0:t-1}|z_{1:t-1})}{p(z_t|z_{1:t-1})}$$

$$= \frac{p(z_t|x_t)\; p(x_t|x_{t-1})}{p(z_t|z_{1:t-1})}\; p(x_{0:t-1} \mid z_{1:t-1})$$

$$\propto\; p(z_t \mid x_t)\, p(x_t \mid x_{t-1})\, p(x_{0:t-1} \mid z_{1:t-1})$$

Sustituyendo la expresión anterior y 3.17 en 3.16, la ecuación de actualización de pesos puede escribirse como:

$$w_t^i \propto \frac{p(z_t \mid x_t^i)\; p(x_t^i \mid x_{t-1}^i)}{q(x_t^i \mid x_{0:t-1}^i,\, z_{1:t})\; q(x_{0:t-1}^i,\, z_{1:t-1})} \;=\; w_{t-1}^i\; \frac{p(z_t \mid x_t^i)\; p(x_t^i \mid x_{t-1}^i)}{q(x_t^i \mid x_{0:t-1}^i,\, z_{1:t})} \tag{3.18}$$

Además, si $q(x_t \mid x_{0:t-1},\, z_{1:t})$=$q(x_t \mid x_{t-1},\, z_t)$, entonces la densidad de importancia se vuelve sólo dependiente de x_{t-1} y z_t. Esto es particularmente útil en el caso común cuando sólo una estimación filtrada de $p(x_t \mid z_{1:t})$ es requerida a cada instante. A partir de este momento se asumirá tal caso.

El peso modificado es entonces:

$$w_t^i = \; w_{t-1}^i \frac{p(z_t \mid x_t^i)\; p(x_t^i \mid x_{t-1}^i)}{q(x_t^i \mid x_{t-1}^i,\, z_{1:t})} \tag{3.19}$$

y la densidad posterior filtrada $p(x_t \mid z_{1:t})$ puede ser aproximada como:

$$p(x_t \mid z_{1:t}) \approx \sum_{i=1}^{N_s} w^i \delta\,(x - x_t^i) \tag{3.20}$$

donde los pesos están definidos en 3.19. Puede demostrarse que si $N_s \to \infty$ la aproximación 3.20 se acerca a la verdadera densidad posterior $p(x_t \mid z_{1:t})$.

El algoritmo SIS por tanto consiste en una pequeña propagación recursiva de los pesos y las muestras tal como cada medición es recibida de manera secuencial. Una descripción en pseudo-código se ve en el algoritmo 3.1:

Algorithm 3.1: Algoritmo del filtro de partículas SIS

for $i = 1$ *hasta* N_s **do**
    Esboza $x_t^i \sim q(x_{t-1}^i \mid z_t)$
    Pondera partícula, w_k^i de acuerdo a 3.19
end for

El problema del empobrecimiento de la partícula

Un problema común con el filtro de partículas SIS es el fenómeno de la degeneración o empobrecimiento. Después de varias iteraciones todas excepto una partícula tendrán un peso despreciable. Se ha demostrado que la varianza de las ponderaciones de importancia pueden sólo incrementarse con el tiempo, y por tanto es imposible evitar éste fenómeno. Este empobrecimiento implica un gran esfuerzo computacional, dedicado a la actualización de partículas cuya contribución es la aproximación de $p(x_t \mid z_{1:t})$, es casi cero. Una medición aceptable de la degeneración del algoritmo es el tamaño efectivo de la muestra N_{eff}, y se define como:

$$N_{eff} = \frac{N_s}{1 + Var(w_t^{*i})^t} \tag{3.21}$$

donde $w_t^{*i} = p(x_t^i \mid z_{1:t}) \,/\, q(x_t^i \mid x_{t-1}^i,\, z_t)$ es conocida como el *"el verdadero peso"*. Como esto no puede ser evaluado exactamente, se hace una estimación $\widehat{N_{eff}}$ de N_{eff} que se obtiene con:

$$\widehat{N_{eff}} = \frac{1}{\sum_{i=1}^{N_s} (w_t^i)^2} \tag{3.22}$$

donde w_{t^i} es el peso normalizado obtenido con 3.18. Se advierte que $N_{eff} \leq N_s$ y un N_{eff} pequeño indica un empobrecimiento severo. Claramente, el problema de la degeneración es un efecto indeseado de los filtros de partículas. Hay varios enfoques para evitar este problema:

- Utilizar un N_s muy grande, pero es poco práctico.
- La elección de una buena densidad de importancia que miniminice $Var(w_t^{*i})$. En [12] hacen uso de este enfoque para reducir el numero de partículas necesarias en el problema del SLAM.
- El remuestreo.

Remuestreo

Este método para reducir los efectos del empobrecimiento se basa en usar remuestreos siempre que una degeneración significativa es observada (Cuando N_{eff} llega a ser menor que un umbral,N_T). La idea básica del remuestreo es eliminar partículas que tienen pesos bajos y concentrarse en las partículas con pesos altos. El paso del remuestreo involucra la generación de un nuevo conjunto $\{x_t^{i*}\}_{i=1}^{N_s}$ al remuestrear (con reemplazo) N_s veces a partir de una representación discreta aproximada de $p(x_t \mid z_{1:t})$ dada por:

$$p(x_t \mid z_{1:t}) \approx \sum_{i=1}^{N_s} w^i \delta\ (x - x_t^i) \tag{3.23}$$

de tal forma que $Pr(x_t^{i*} = x_t^j) = w_t^j$. La muestra es de hecho una distribución muestra independiente e idéntica de la distribución discreta 3.23, y por lo tanto los pesos ahora se reasignan a $w_t^i = \frac{1}{N_s}$.

Aunque el paso de remuestreo reduce los efectos del problema de degeneración introduce otros problemas prácticos. Primero, limita la oportunidad de paralelizar el proceso ya que todas las partículas deben ser combinadas. Segundo, las partículas que tienen altos pesos w_t^i son estadísticamente seleccionadas muchas veces. Esto conduce a una pérdida de diversidad entre las partículas y además la muestra resultante contendrá muchos puntos repetidos. Este problema, conocido como *empobrecimiento de la muestra*, se agrava en el caso de que exista ruido en el proceso. De hecho, para el caso de ruidos muy ligeros todas las partículas colapsarán a un sólo punto después de pocas iteraciones.

Filtros de partículas aplicados al SLAM

Como se vió anteriormente, el filtro de partículas es recursivo y opera en dos fase: la *predicción* y la *actualización*. En este aspecto se asemeja mucho a los filtros de Kalman estudiados

en la sección 3.1.1.1. La predicción está en el modelo de movimiento del robot, generando las N partículas. La actualización está en la ponderación de las partículas generadas, utilizando para ello el modelo de observación.

Para resolver el problema del SLAM, cada partícula del filtro de partículas representa una trayectoria posible del robot y un mapa, y no únicamente posiciones, a diferencia de los EKFs [34]. La idea principal es estimar una distribución posterior $p(s_{1:t} \mid z_{1:t}, u_{0:t})$ sobre las trayectorias potenciales $s_{1:t}$ de un robot dadas sus observaciones $z_{1:t}$ y sus mediciones de odometría $u_{0:t}$ y utilizar esta distribución posterior para calcular una distribución posterior de mapas y trayectorias [12]:

$$p(s_{1:t}, m \mid z_{1:t}, u_{0:t}) = p(m \mid s_{1:t}, z_{1:t})\, p(s_{1:t} \mid z_{1:t}, u_{0:t}) \tag{3.24}$$

Esto puede hacerse de manera eficiente ya que la distribución posterior sobre mapas $p(m \mid s_{1:t}, z_{1:t})$ puede calcularse analíticamente, dado por conocidos $s_{1:t}$ y $z_{1:t}$. Y puede hacerse ésto gracias a que las trayectorias y los mapas son condicionalmente independientes [34].

Para estimar la distribución posterior $p(s_{1:t} \mid z_{1:t}, u_{0:t})$ sobre las trayectorias potenciales, se utiliza un filtro de partículas en donde un mapa individual se asocia a cada muestra. Cada mapa es construido dadas las observaciones $z_{1:t}$ y las trayectorias $s_{1:t}$ representadas por la partícula correspondiente.

Capítulo 4

Plataforma de desarrollo

El siguiente capítulo está centrado en el estudio de la plataforma hardware y software sobre la cual se ha desarrollado este proyecto. En primer lugar, se realiza una breve introducción del Sistema Operativo utilizado así como de las herramientas de programación utilizadas. En segundo lugar, se describen algunos de los Frameworks [1] existentes en la actualidad, resaltando tanto la plataforma ROS como el simulador Player/stage, herramientas fundamentales para conseguir los objetivos propuestos al comienzo del proyecto. Se finaliza el capítulo, con una descripción del manipulador autónomo Manfred con el que se ha trabajado a lo largo de la investigación.

4.1. Sistema Operativo. Linux

El entorno elegido para desarrollar este proyecto ha sido Linux por tratarse de un sistema operativo de gran estabilidad y compatible con todas las aplicaciones necesarias. Además es el sistema operativo instalado en Manfred robot con el cual trabajaremos y del que hablaremos más adelante, a lo que se une el hecho de que la versión más estable de ROS se encuentra en Linux.

[1] Al hablar de marcos de trabajo es habitual utilizar el término inglés Framework, expresión que emplearemos en este documento.

En concreto se ha optado por Ubuntu, que se trata de una distribución GNU/Linux, cuya licencia es libre y de código abierto. Otras ventajas de este sistema operativo son:

- Multitarea, multiplataforma, multiusuario y multiprocesador.
- Protección de la memoria, haciéndolo más estable frente a caídas del sistema.
- Carga selectiva de programas según la necesidad.
- Uso de bibliotecas enlazadas tanto estáticamente como dinámicamente.

4.2. Herramientas de Programación.

4.2.1. C++

Parte del código utilizado en el proyecto ha sido descargado del repositorio de ROS, el cual está escrito en C++, motivo por el cual se ha elegido este lenguaje de programación.

C++ es un lenguaje de programación de propósito general basado en el lenguaje de programación C. El lenguaje C nació en los laboratorios Bell de AT&T y ha sido asociado con el sistema operativo UNIX, su eficiencia y claridad han hecho que el lenguaje ensamblador apenas haya sido utilizado en UNIX. Desde 1980 el lenguaje C ha ido evolucionando añadiendo características como clases, chequeo del tipo de argumentos de una función y conversión, así como otras características; dando como resultado el lenguaje llamado *C con Clases*.

En 1983, *C con Clases* fue rediseñado, extendido y nuevamente implementado. El resultado se denominó *Lenguaje C++*. Las extensiones principales fueron *funciones virtuales*, *funciones sobrecargadas* (un mismo identificador puede representar distintas funciones), y *operadores sobrecargados* (un mismo operador puede utilizarse en distintos contextos y con distintos significados), tras alguna otra mejora C++ quedó disponible en 1985.

Desde entonces, C++ ha sido revisado y mejorado, añadiendo nuevas características, como herencia múltiple, funciones miembro **static** y **const**, miembros **protected**, plantillas referidas a tipos y manipulación de excepiones. C++ ha conseguido ser un lenguaje potente, eficiente y

seguro, caracteríticas que lo convierten en un estándar dentro de los lenguajes de programación orientada a objetos. Destacando por:

- Programación orientada a objetos.
- Portabilidad.
- Brevedad.
- Programación modular.
- Compatibilidad con C.
- Velocidad.

Recomendamos consutar [29] si se quieren adquirir más conocimientos acerca de la programación C++.

4.2.2. OpenCV

La herramienta de programación OpenCV ha sido empleada en la fase inicial de nuestro proyecto, en el apéndice B explicamos con más detalle como ha sido utilizada.

"OpenCV" es una biblioteca libre de visión artificial originalmente desarrollada por INTEL. Su primera versión alfa apareció en enero de 1999, y desde entonces se ha utilizado en infinidad de aplicaciones como por ejemplo sistemas de seguridad con detección de movimiento, aplicaciones de control de procesos donde se requiere reconocimiento de objetos, etc. El hecho de que su publicación se de bajo licencia BSD, permite que sea usada libremente para propósitos comerciales y de investigación con las condiciones en ella expresadas.

Open CV es multiplataforma, existiendo versiones para GNU/Linux, Mac OS X y Windows; contiene más de 500 funciones que abarcan una gran gama de áreas en el proceso de visión, como reconocimiento de objetos (reconocimiento facial), calibración de cámaras, visión estéreo y visión robótica.

El proyecto pretende proporcionar un entorno de desarrollo fácil de utilizar y altamente eficiente. Esto se ha logrado, realizando su programación en código C y C++ optimizados,

aprovechando además las capacidades que proveen los procesadores multi núcleo. Algunas de las aplicaciones más conocidas son las sguientes:

- OpenCV ha sido usada en el sistema de visión del vehículo no tripulado Stanley de la Universidad de Stanford, ganador en el año 2005 del Gran desafío DARPA.
- OpenCV se usa en sistemas de vigilancia de vídeo.
- OpenCV es la clave en el programa Swistrack, una herramienta de seguimiento distribuida.

Toda la informacion sobre OpenCV puede consultarse en la siguiente dirección:
`http://opencv.willowgarage.com/wiki/`.

4.2.3. OpenSlam

Las librerías "Gmapping" y "Tynislam" utilizadas en el proyecto pertenecen al proyecto OpenSLam y ambas implementan técnicas de SLAM.

El problema de la Localización y Modelado Simultáneo (SLAM) comenzó a estudiarse hace años siendo varias las técnicas propuestas, no obstante pocas de estas implementaciones han sido puestas a disposición de la comunidad científica. Es por esto, por lo que surgue la iniciativa OpenSLam, una colección de paquetes SLAM de código abierto disponible en el sitio web Open SLAM [21] desde enero de 2007.

Su objetivo es proporcionar una plataforma para los investigadores SLAM en la cual puedan publicar sus algoritmos, para ello ofrecen un servidor (svn) y una página web en los cuales pueden difundir y promocionar su trabajo. En el repositorio, sólo los autores tienen acceso a los archivos, el resto de ususarios tienen limitado el acceso a sólo lectura.

OpenSlam no obliga a los autores a ceder los derechos de autor de su código, sólo requiere que se proporcione el código fuente de los algoritmos y que se permita a los usuarios usar y modificar dicho código en beneficio de su propia investigación. Cualquier aplicación comercial, redistribución, etc, tiene que ser negociado entre los usuarios y los autores de forma particular.

4.2.4. RADISH: The Robotics Data Set Repository

Hemos acudido al repositorio de Radish [26] para obtener registros de datos de odometría y láser de robot reales que nos permitiesen simular los algoritmos con los que hemos trabajado: CoreSlam y Gmapping.

"Radish"es una iniciativa estadounidense creada por Andrew Howard y Nick Roy que pretende ser un repositorio de datos estándar para la comunidad robótica. Nació en mayo del 2003 con el objetivo de facilitar el desarrollo, evaluación y comparación de algunos de los algoritmos utilizados en la robótica. El enfoque actual está centrado en la localización y mapeado, aunque se espera que pueda extenderse a otros ámbitos de la robótica.

Los investigadores pueden descargarse y hacer uso de los conjuntos de datos disponibes así como hacer sus propias contribuciones al repositorio. El nuevo conjunto de datos puede ser enviado a través de una página de presentación y los suscriptores reciben por correo noticias y notificaciones de la subida de archivos nuevos.

Se acepta cualquier formato de datos siempre que éste esté bien documentado, aunque el formato preferido es el de CARMEN (en la sigiente sección se hablará de Carmen). En la actualidad, el repositorio Radish contiene una colección de 39 conjuntos de datos que se agrupan en cuatro categorías:

- Registros de datos de odometría, láser y sonar tomados de robots reales.
- Registros de datos de todo tipo sensores tomados de robots reales.
- Mapas de entorno generados por robots.
- Mapas de entorno generados a mano.

4.3. Frameworks

Desarrollar un robot autónomo es altamente complicado. Afortunadamente, en la actualidad es posible encontrar diversas soluciones comerciales o proyectos de código abierto, centrados en la programación de robots, los cuales ofrecen facilidades en esta complicada tarea. Gracias a los módulos y a las librerías proporcionadas por alguna de estas soluciones, la programación de aspectos de bajo nivel pasa a un segundo plano.

Cuando se trabaja en un proyecto de robótica, una tarea típica consiste en desarrollar experimentos simulados previos a la implementación real. Esta búsqueda de ahorro de dinero usualmente se torna costosa en tiempo por la generación del entorno simulado y de los robots a utilizar, intentando que tengan una fidelidad adecuada comparada con la realidad en donde se esperan implementar. Una vez que se obtienen buenos resultados del proceso de simulación, avanzar a la realidad requiere de otra etapa consumidora de tiempo al preparar y adecuar los códigos y algoritmos a las conexiones reales del Robot.

Como consecuencia, en la comunidad robótica comenzó a crecer la inquietud por desarrollar un mejor software que permitiera una mayor transparencia entre los experimentos simulados y las implementaciones reales, con la idea de reducir los tiempos de desarrollo al mismo tiempo que se pudieran lograr soluciones relevantes, eficientes y robustas. Apostaron por soluciones robóticas capaces de integrar los conceptos de: modularidad, reutilización de código, escalabilidad, integración y abstracción de hardware; así como aplicaciones cuyo mantenimiento fuese simple y rápido y en el que las actualizaciónes fuesen fáciles.

Uno de los primeros paquetes que consiguió incorporar estos conceptos fue el proyecto Player [23] nacido en 2001. La reutilización de códigos que permite este software y sus capacidades de rápida simulación en 2D hicieron de éste un popular sistema de contribuciones científicas. Posteriormente, en el año 2006 salía la primera versión de lo que hoy se conoce como Microsoft Robotics Studio [28], robótica orientada a servicios, a pesar de ser software libre y ofrecer servicios de simulación 3D, su carácter de código cerrado hizo que no alcanzase el exito esperado.

Al año siguiente y confirmando la tendencia de hacer software bajo los conceptos ya mencionados nacía el proyecto ORCA [22], como un software orientado a componenetes en donde drivers y algoritmos se volvían bloques de programación denominados componentes, simplificando aún más el desarrollo de aplicaciones complejas a partir de la integración de partes simples.

En 2008 se crea ROS [39], como una evolución al proyecto Player y ORCA, ofreciendo una arquitectura y funcionalidad orientada a servicios con carácter de software abierto y con base Linux. ROS es hoy por hoy quizá el más popular software de diseño de proyectos y soluciones robóticas. Parte de nuestra labor en este proyecto ha consistido en familiarizarnos con esta herramienta de programación.

Todos los softwares han promovido la generación de soluciones robóticas orientadas a servicios, preveyendo el camino para el rápido prototipado en simulación e implementación real de tal forma que ya existen disponibles gran cantidad de servicios reusables e integrables.

4.3.1. Carmen

Dedicamos este apartado al software Carmen pues ha sido necesario instalarlo como requisito indispensable para poder compilar el módulo Gmapping perteneciente a ROS.

CARMEN, “Carnegie Mellon Navigation”, es una colección de código abierto de software para el control de robots moviles diseñada en el año 2002 con el fin de proporcionar una interfaz coherente y un conjunto de herramientas básicas para la investigación robótica aplicable a una amplia variedad de plataformas de robots comerciales. Es un software modular diseñado para proporcionar primitivas básicas de navegación, incluyendo: control de sensores, evasión de obstáculos, localización, planificación de rutas y mapeado. Algunas de sus funcionalidades principales son las siguientes:

- Software modular.
- Utilización de una plataforma IPC para la comunicación entre procesos.
- Proceso de monitoreo.

- Soporte Hardware para diferentes plataformas.
- Simulador robot/sensor (en 2D).
- Módulo de ruta de planificación.
- Módulo de localización.
- Módulo de scan-matching y mapping.
- Registro de mensajes y funcionalidad de reproducción.
- Servidor de parámetros centralizado.
- Varias funciones útiles para trabajar o programar robots.
- Aunque está escrito en C, proporciona soporte para JAVA.
- Se ejecuta en LINUX y está disponible bajo licencia GPL (Gnu Public License).

El proyecto cuenta con su propia página web a disposición de todos los usuarios: `http://carmen.sourceforge.net/`,(Accedido en septiembre 2011).

4.3.2. Player/Stage

Durante el proyecto se ha trabajado con entornos simulados para los cuales se ha requerido el uso de las herramientas ofrecidas por la plataforma Player/Stage. Trabajar con simulación nos ha permitido ejecutar el programa como si del propio robot se tratara aunque teniendo en cuenta que las condiciones del entorno simulado no son las mismas que las del entorno real, (se explicará este concepto con más detalle en el capítulo **??**). Nos decidimos por Player/Stage por ser compatible con ROS, lo que nos ha facilitado la programación.

El proyecto *Player/Stage* [23] fue fundado por Brian Gerkey (Stanford), Richard Vaughan (SFU) y Andrew Howard (NASA JPL) en el año 2000, basándose en software que habían desarrollado con Kasper Stoy y otros en los Laboratorios de Investigación Robótica de la Universidad de California del Sur. El nombre del proyecto fue tomado de una célebre frase de Shakespeare *"All the world's a stage, And all the men and women merely players"* (El mundo entero es un escenario, y los hombre y mujeres son simplemente actores). El proyecto está compuesto por tres componentes bien diferenciados:

- *Player*, un servidor de dispositivos robóticos.
- *Stage*, un simulador multi-robot en 2D.
- *Gazebo*, un simulador multi-robot en 3D.

A continuación se explicarán los componentes *Player* y *Stage* únicamente, puesto que el componente Gazebo no ha sido utilizado en este proyecto.

Player

Player es un servidor en red multihilo para el control de robots. Ejecutándose en un robot, *Player* proporciona una interfaz simple y clara para comunicarse con los sensores y actuadores del robot a través de las red IP. El programa cliente habla con *Player* a través de un *socket* TCP, leyendo datos de los sensores, escribiendo órdenes en los actuadores y configurando dispositivos.

La plataforma original de *Player* es la familia ActivMedia Pioneer 2, pero muchos otros robots y sensores comunes son soportados actualmente. La arquitectura modular *Player* hace que sea muy fácil añadir soporte nuevo hardware, además existe una comunidad de ususarios y desarrolladores que contribuyen con nuevos *drivers*.

Player se puede ejecutar en la mayoría de las plataformas compatibles con POSIX, incluyendo sistemas empotrados. Los requisitos que necesita *Player* son:

- Entorno de desarrollo POSIX.
- Pila de protocolos TCP/IP.
- Una versión reciente de GNU gcc, con soporte para C y C++

Player hace una clara distinción entre la interfaz de programación y la estructura de control, optando por una interfaz de programación general, con la creencia de que los usuarios desarrollarán sus propias herramientas para construir sistemas de control. Es decir, está diseñado para ser independiente del lenguaje de programación y la plataforma.

Un programa cliente puede residir en cualquier máquina que tenga una conexión TCP con el robot en el que se ejecuta *Player*, y puede ser escrito en cualquier lenguaje que soporte *sockets* TCP. Actualmente, existen clientes en C++, Tcl, Java y Python. Además, *Player* no impone restricciones en cuanto a cómo estructurar los programas de control del robot. Por último decir, que *Player* es un código abierto, bajo licencia pública GNU.

Stage

Stage es un entorno de simulación para la plataforma *Player*. Ofrece una interfaz gráfica en dos dimensiones, que imita el mundo real gracias al uso de una imagen que sirve de mapa plano del entorno del robot (Figura4.1).

El entorno de simulación *Stage* funciona como un plugin de *Player*, es decir, como un añadido

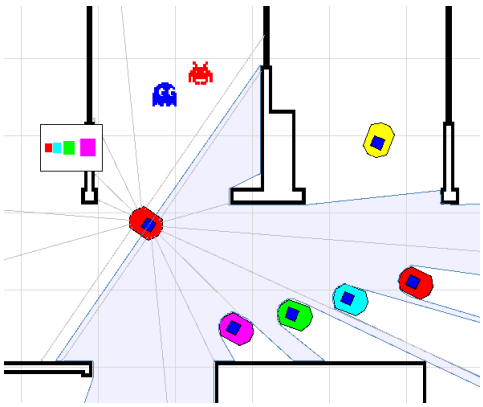


Figura 4.1: *Simulador de robots Stage*

que ofrece nuevas posibilidades. A parte de la interfaz gráfica, ofrece una completa simulación de los sensores o dispositivos de un robot.

Tipos de archivos importantes

En *Player/Stage* hay tres tipos de archivos:

- un archivo **.world**
- un archivo **.cfg** (configuración)
- un archivo **.inc** (incluir)

Lo primero que se debe hacer es definir el fichero *.world* en el cual deben especificarse:

- Los modelos (o robots) que se van a simular.
- Los dispositivos que dichos modelos van a poseer.
- La ventana de interfaz gráfica.
- La imagen que sirve como mapa del entorno.

Una vez se ha definido correctamente el fichero *.world*, sólo queda definir el fichero de configuración de *Player* en el que se especifican las asociaciones entre drivers e interfaces. En el apéndice A podemos consultar un ejemplo sencillo de un archivo *.world*.

4.3.3. ROS

El capítulo cinco estará dedicado exclusivamente a explicar la plataforma ROS, pero cómo primera aproximación podríamos definir a ROS como un software enlatado Building Blocks (librerías, aplicaciones) o sistema de paquetería; que posee diferentes herramientas de desarrollo así como herramientas de inspección y depuración. Se podría resumir en los siguientes tres conceptos:

- Meta-Sistema Operativo.
- (RSF) Framework para el desarrollo de sistemas robóticos.
- Arquitectura robótica desplegada en múltiples máquinas.

Comunidad ROS

ROS es una federación de repositorios con software para robots, que en poco tiempo ha conseguido crecer a un ritmo vertiginoso, ya en el 2010 contaba con:

- Casi 200 Stacks.
- Más de 100 Packages.
- Unos 50 Repositorios Federados.

Consiguiendo convertirse en una wiki unificada y uniforme, que tiene como uno de sus pilares fundamentales la importancia de la integración sobre el desarrollo. Desde su creación en 2007 su evolución ha sido exponencial como se refleja en la siguiente gráica (Figura 4.2).

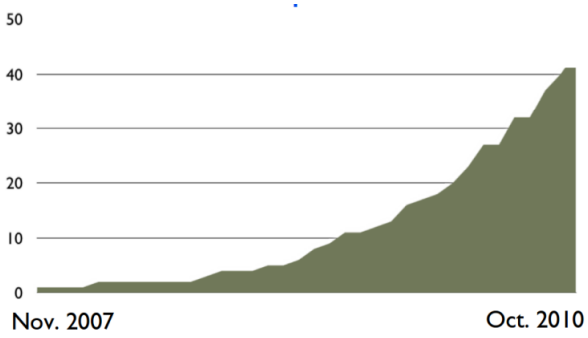


Figura 4.2: *Evolución de ROS.*

¿Por qué usar ROS?

Cada año más de 150 ingenieros se unen al proyecto ROS, el ritmo de crecimiento de las fuentes de líneas de código que están disponibles es imparable: cpp (67.69 %), python (23.57 %), xml (5.26 %), sh (2.05 %), lisp (1.16 %). Destaca por ser un código abierto bajo licencia BSD, que cuenta con una amplia base de documentación, tutoriales y soporte técnico. La fuerza de ROS se ve reflejada en la capacidad de hacer frente a diferentes aplicaciones del ámbito de la robótica.

- Visualización.
- Reconocimiento de objetos.
- Navegación.
- Manipulación.

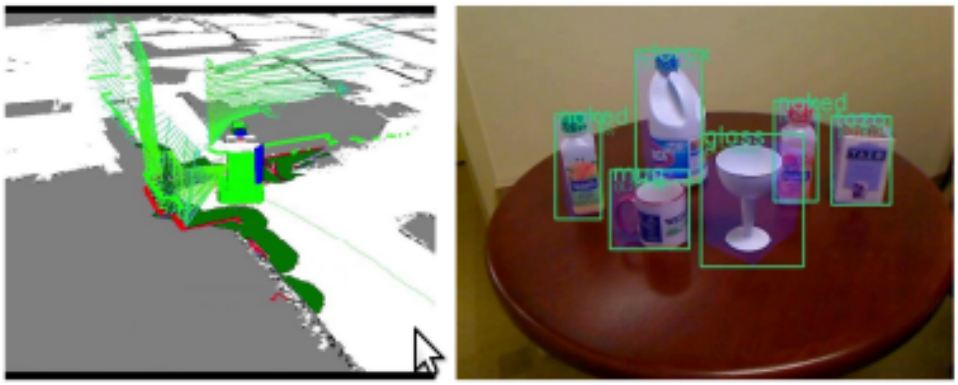


Figura 4.3: *Visualización y reconocimiento de objetos.*

Pero como todo, tienen sus ventajas y sus inconvenientes.

VENTAJAS :

- Reduce el tiempo invertido en infraestructura y se centra en la investigación.
- Aborda problemas de alto nivel.
- Acelera el aprendizaje.
  - Viendo código de otros.
  - Viendo documentación de otros.
- Fomenta el trabajo en equipo y establece convenios, procesos y metodologías para hacer software reusable.

DESVENTAJAS :

- Sus objetivos son muy ambiciosos.
  - Tratan con software muy variado (librerias hechas según criterio del autor).
- La integración de las aplicaciones no es inmediato.
  - Hay que leer suficiente documentación para comprender el funcionamiento.
  - Se debe revisar el código hecho, pues cada aplicación es distinta.
  - Es necesaria una etapa de depuración.

- Curva de aprendizaje.
- Exponer una librería en ROS no es gratis.
- Contínuos cambios y evolución, que pueden dar como resultado un software obsoleto.
- Por ahora no tiene soporte para Windows u otros sistemas empotrados.

Conclusión

Escogeremos ROS si queremos llegar más alla en nuestros proyectos ya que existe una metodología y unos convenios definidos para el desarrollo de software en robótica que nos facilitará nuestra labor. Por otro lado, un aspecto muy importante y que está adquiriendo cada vez mayor importancia en esta sociedad de continuos cambios, es la posibilidad de fomentar la interacción con otros grupos de investigación, ROS apoya esta iniciativa. En la actualidad más de 50 robots usan ROS.

Figura 4.4: *Ejemplos de robots que usan ROS.*

4.4. Manfred

Manfred (Figura 4.5) es un manipulador autónomo antropomórfico, avanzado, fiable y seguro. Nació en el año 2001 convirtiéndose en un robot pionero en España, durante sus dos primeros años de vida los investigadores se dedicarón a la construcción mecánica, especialmente al brazo. Una vez finalizado su diseño y construcción en el año 2003, se comenzó a trabajar en diferentes ámbitos, más ligados a los aspectos más inteligentes del robot, como el aprendizaje del entorno y la localización.

Figura 4.5: *Robot MANFRED*

Manfred está diseñado para ser capaz de desenvolverse de forma autónoma en entornos típicamente humanos, es decir, pasillos, salas, despachos donde en ocasiones resulta necesario abrir y atravesar puertas, evitar obstáculos o coger y manipular objetos,etc. Todo esto requiere que el robot integre los siguientes aspectos:

1. Todas las capacidades básicas de un robot móvil para desplazarse de forma segura y autónoma por el entorno.

2. La coordinación motora entre la base y el manipulador.

3. La coordinación sensorial necesaria para poder manipular objetos.

Todo sistema robótico está formado por una serie de subsistemas que posibilitan, mediante su interconexión, el cumplimiento de los objetivos para los cuales ha sido diseñado el robot. Esta serie de módulos utilizan la información del entorno de trabajo para generar datos que son utilizados al final para dotar de capacidad de movimiento, tanto al brazo manipulador como a la base móvil. A continuación se describen con más detalle los principales componentes de los sistemas que constituyen el manipulador móvil.

4.4.1. Estructura del robot.

Manfred está formado por una estructura metálica que permite integrar todos los componenetes que necesita para su funcionamiento y que se puede dividir en tres partes:

1 Base móvil. La base está constituida por dos plataformas de acero con un diámetro de 61 cm y una altura en torno a 65 cm. Está equipada con ruedas que le permiten desplazarse, en la base se aloja el sistema de baterías que generan la alimentación necesaria para que el robot pueda funcionar de manera autónoma.

2 Torso fijo. Sobre la base móvil se ha montado una estructura (bastidor) que hace de cuerpo del robot y que aloja: todo el cableado para la conexión del brazo manipulador con el ordenador, el cableado para la transmisión de potencia desde la base a los motores del brazo, los servoamplificadores asociados con los motores del manipulador y todo el cableado correspondiente a los sensores externos. Esta estructura sirve también como soporte para el anclanaje del brazo, el sensor láser y de las cámaras para el sistema de visión.

3 Brazo manipulador. El brazo manipulador constituye el elemento fundamental del robot manipulador Manfred. Compuesto por elementos rígidos conectados por medio de articulaciones de revolución. Cada articulación conforma un grado de libertad hasta alcanzar un total de seis. Manfred posee una gran flexibilidad para realizar tareas de agarre y desplazamientos de objetos, combinando los diversos grados de libertad de los que dispone.

4.4.2. Sistema sensorial.

El sistema sensorial permite transformar las variables físicas que caracterizan el entorno en un conjunto de datos que serán procesados por otros módulos como por ejemplo los encargados de la localización, el planificador de movimiento, etc dando como resultado acciones de control que gobiernan el comportamiento del robot. Este sistema consta de diversos elementos:

1 Subsistema de telemetría láser. Su objetivo es proporcionar al robot la información necesaria sobre la distancia a los objetos del entorno para realizar un modelado del espacio de trabajo que rodea al manipulador móvil. Esta información se utiliza principalmente en las tareas de navegación y localización del robot durante sus desplazamientos en un espacio de trabajo amplio. Este subsistema consta a su vez de los siguientes elementos:

1. Telémetro láser Hokuyo UTM-30LX de 270 grados de apertura, situado en la parte posterior del vehículo. Tiene un rango de distancia de detección de 100mm hasta 30,000mm (30m) y un período de escaneo mínimo de 25msec y cuenta con una resolución de 0.25º. Se conecta al ordenador mediante una interfaz USB2.0. Su consumo de potencia es 12V y 700mA lo que le hace apto para ser utilizado en sistemas alimentados con baterías como es nuestro caso.

2. Telémetro láser bidimensional PLS de Sick de 180 grados de apertura que permite la adquisición de datos en 2D. Para lograr la portabilidad a tres dimensiones está instalado sobre un chasis motorizado que permite su movimiento vertical dentro de un rango de 45 grados. Durante la fase de navegación en entornos poco ocupados, como pueden ser pasillos, se utiliza telemetría 2D de forma que el escáner realice un barrido en un plano paralelo al suelo.

   Este sensor está preparado para capturar 361 medidas en un barrido horizontal de 180 grados (se toman medidas cada 0.5º). El PLS garantiza unos errores en la medida de profundidad que no superan los 20 mm. Este error se ve influenciado por dos parámetros, la distancia a medir y el ángulo de disparo del haz láser (de 0º a 180º)(Figura 4.6).

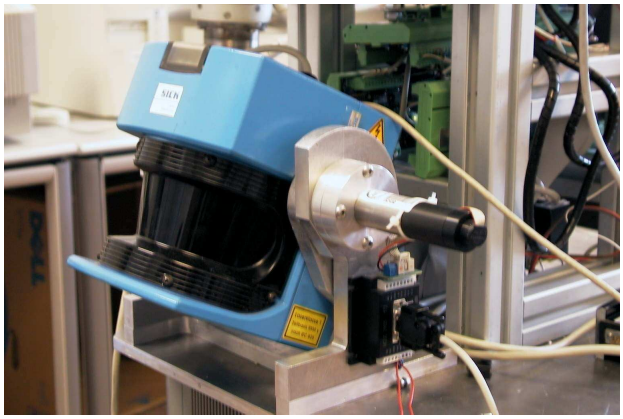


Figura 4.6: *Telémetro láser PLS-220 de la casa SICK*

2 Subsistema de visión. Se pretende que el robot sea capaz de manipular objetos situados en un entorno 3D por lo que es necesario reconocer el objeto a manipular, determinar su posición y orientación relativa con respecto al manipulador móvil, así como determinar el punto y orientación adecuado para su manipulación. Las tareas de manipulación previstas consisten en la apertura de puertas y paso a través de las mismas, pulsar interruptores o coger objetos simples de una mesa. Para llevar a cabo estas tareas el subsistema de visión consta de los siguientes elementos:

a) Cámaras de color. El sistema incorpora una cámara SONY EVI-D100 para reconocer los objetos y estimar su posición respecto al robot,esta cámara se sitúa en la parte frontal del cuerpo del manipulador móvil. El robot también incorpora una minicámara SONY B/N XC-ES50CE situada sobre la muñeca del manipulador que se utiliza en las tareas de manipulación cuando el brazo se encuentra situado delante del objeto a manipular.

b) Cámara 3D. Manfred incorpora una cámara de tecnología time-of-flight que le permite obtener una imagen 3D que consiste en una matriz de distancias a los distintos objetos que se encuentra delante. La información de la cámara 3D y la cámara de color convencional se fusiona para poder realizar una segmentación de objetos más precisa, facilitando así la manipulación de éstos.

3 Sensor fuerza/par. Para la interacción con el entorno en las tareas de manipulación el robot Manfred porta en el extremo del brazo un sensor fuerza-par JR3 modelo 67M25A-U560. El objeto principal de la utilización de este sensor es poder realizar tareas de manipulación basadas en control de fuerza o par, como por ejemplo apertura de puertas, pulsación de interruptores, recogida de objetos, etc.

4 Sensores cinemáticos. La principal función de los sensores cinemáticos es aportar información sobre la posición, pudiendo obtener en el sistema de control las variables derivadas (velocidad y aceleración). Estos sensores se encuentran principalmente acoplados a los ejes de los motores proporcionando información del giro de éstos en forma de cuentas de encoder. Esta información puede ser interpretada de dos formas, se obtiene la posición relativa o absoluta del sistema acoplado a dicho motor así como sus variables derivadas.

4.4.3. Sistema locomotor.

El sistema locomotor del manipulador móvil está constituido por la base móvil citada anteriormente, a la que se incorporan cinco ruedas: tres ruedas de apoyo que mejoran la estabilidad del robot y facilitan su movimiento, y dos ruedas motrices asociadas con motores sin escobillas y sus correspondientes servoamplificadores. La ruedas motrices dan lugar a un desplazamiento de tipo diferencial que permite al robot girar sobre si mismo.

4.4.4. Sistema de alimentación.

La base móvil alberga el sistema de alimentación compuesto por las baterías que dotan de autonomía a todo el robot. Consiste una conexión serie de cuatro baterías de 12 V para proporcionar una tensión de alimentación de 48 V en continua. Las baterías seleccionadas son el modelo LC-X1242P de PANASONIC que proporcionan una tensión de salida de 12V y una capacidad de 42 A/h.

Además, como sistema de seguridad, lleva instalado un sistema de monitorización de las baterías a través de un micro-controlador PIC16F818, que permite medir en todo instante la tensión proporcionada por las baterías y la corriente que circula por ellas. Este sistema permite comunicar continuamente el estado de la alimentación al ordenador de control, así como realizar una parada automática controlada de los motores del robot en caso de baja tensión de alimentación o de superar una corriente umbral.

4.4.5. Sistema manipulador LWR-UC3M-1.

Con objeto de realizar las tareas de manipulación, Manfred está equipado con un brazo robótico ligero de seis grados de libertad el cual ha sido íntegramente diseñado en la Universidad Carlos III de Madrid. Las características principales de este brazo manipulador son:

a) Redundancia cinemática similar al brazo humano.

b) Masa del conjunto de 18 kilogramos.

c) Capacidad de carga máxima de 4.5 kilogramos en el extremo del brazo.

d) Relación carga/peso entre 1:3 y 1:4.

e) Alcance en torno a 955 milímetros.

Dicho brazo se monta lateralmente de forma que el sistema de visión y la telemetría láser 3D puedan percibir lo que se desea manipular sin ser obstaculizados por el brazo. Las articulaciones del brazo están compuestas por motores sin escobillas “Brushless” de corriente continua que permiten alimentar directamente en contínua a los accionadores, una etapa de reducción de velocidad y aumento de par de tipo Harmonic Drive, más concretamente de la compañía alemana Harmonic Drive AG en su categoría HFUC-2UH que incorporan rodamientos de salida integrados.

4.4.6. Sistema de procesamiento.

Un manipulador móvil es un sistema de enorme complejidad que debe ser capaz de procesar e interpretar una gran cantidad de información sensorial y cerrar varios lazos de control simultáneamente. Esta complejidad se traduce directamente en el crecimiento y complejidad del software que se desarrolla.

Para favorecer el desarrollo y mantenimiento del sistema software del robot se ha optado por una arquitectura completamente modular que traduce cada función a realizar por el sistema en un módulo software independiente. Para ello, se han desarrollado módulos funcionales que puedan trabajar conjuntamente mediante el intercambio de datos pero que se ejecuten como procesos completamente independientes. Con esta estructura modular se facilita el desarrollo, comprobación y actualización de los diferentes algoritmos.

Cada uno de los módulos funcionales se diseña con capacidad para el auto-diagnóstico, de esta forma se dispone de un primer control de fallos en cada uno de los módulos. Al tratarse de módulos de ejecución independientes, cada tarea o cada fase de una tarea definirá el conjunto de módulos necesarios para su desarrollo. El computador asignado al sistema de control coordinado se encarga de las funciones básicas de control del sistema y lleva incorporado una tarjeta controladora de ocho ejes con la que se comunica vía memoria de doble puerto. Dicha tarjeta permite cambiar dinámicamente los parámetros y el programa de control que se ejecuta para disponer de un sistema flexible que implemente técnicas avanzadas de control. Esta tarjeta realiza el control de la base (2 g.d.l) y del brazo (6 g.d.l).

Capítulo 5

ROS

Este capítulo está dedicado íntegramente a la plataforma ROS. Se explicarán los conceptos necesarios para entender su funcionamiento y se resumirán las aplicaciones que han sido utilizadas. El objetivo es que el lector se familiarice y llegue a comprender el funcionamiento de la herramienta de programación con la que se ha llevado a cabo este trabajo.

5.1. Introducción.

Robot Operating System (ROS) Plataforma software para el desarrollo software de robots. Originalmente fue desarrollada en 2007 bajo el nombre *switchyard* (subestación), por el Laboratorio de Inteligencia Artificial de Standford en apoyo al proyecto Standford AI Robot (STAIR) [27][24]. A partir del 2008, el desarrollo continúa en el instituto de investigación de Willow Garage [25].

Además de "Robot Operating System", sus siglas también adquieren el significado de "Robot Open Software" ya que el software es completamente abierto y está disponible en `http://www.code.ros.org/gf/` (Accedido en septiembre 2011), para que sea utilizado libremente por cualquiera, modificado o comercializado junto con otros productos.

En el capítulo anterior ya vimos que ROS puede clasificarse como un Framework de Desarrollo de Robótica (RSF), ofreciendo herramientas y librerías para el desarrollo de sistemas robóticos. A pesar de su nombre, ROS no es un sistema operativo propiamente dicho (de hecho funciona sobre Linux), sino una infraestructura de desarrollo, despliegue y ejecución de sistemas robóticos, además de ser una gran federación de repositorios que ofrecen paquetes con software de todo tipo para robots.

A diferencia de la mayoría de los RSFs existentes, ROS pretende dar una solución integral al problema de desarrollo de robots aportando soluciones a las áreas de: Abstracción de Hardware (HAL) y reutilización e integración de robots y dispositivo,s para lo cual encapsula éstos tras interfaces estables y mantiene las diferencias entre los archivos de configuración.

¿Quién está detrás de todo esto?

Se trata de un proyecto de Software Libre que cuenta con el respaldo de una empresa llamada Willow Garage, aspecto muy importante ya que el apoyo de empresas en proyectos de software libre fomenta la promoción del mismo, dos ejemplos claros son Eclipse y Ubuntu. Además, Willow Garage cuenta con el apoyo y colaboración de grupos de investigación como STAIR y profesores reconocidos como Gary Bradsky (principal responsable del proyecto de visión artificial OpenCV).

Por último, mencionar a Brian Gerkey, uno de los principales responsables del proyecto Player (probablemente el RSF más reconocido hasta la fecha), él actualmente forma parte del equipo de ROS. Todos estos factores son una garantía debido a que Standford es una de las cunas de la informática y la robótica, donde han nacido empresas tan importantes como Apple y Google.

5.2. Objetivo del diseño

Cada vez es más frecuente el empleo de RO. Universidades de América, Japón y Europa lo utilizan en el desarrollo de sus investigaciones. Su filosofía de trabajo podemos resumirla en los siguientes cinco puntos:

- Peer to Peer.
- Ofrecer numerosas herramientas.
- Multi-lenguaje.
- Modular.
- Código libre y abierto.

A. Peer-to-Peer

Un sistema construido con ROS consiste en una serie de procesos que pueden encontrarse en máquinas diferentes, y que permanecen conectados en tiempo de ejecución mediante una topología Peer-to-Peer. Si los equipos de trabajo están conectados a una red heterogénea, no es necesario un servidor central para conseguir que todas las máquinas en servicio desarrollen los cálculos que requieren cada una de las tareas (Figura 5.1).

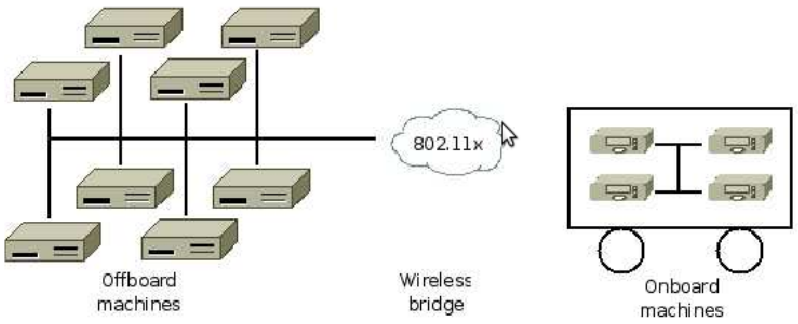


Figura 5.1: *Configuración típica de una red ROS.*

B. Multi-lenguaje

ROS es capaz de trabajar en diferentes lenguajes de programación: C++, Python, Octave y LISP. En lugar de proporcionar una aplicación basada en C con interfaces de código auxiliar para la mayoría de los lenguajes, implementa éstas de forma nativa en cada lenguaje para seguir mejor las convenciones de cada uno de ellos.

En este proyecto en concreto, hemos trabajado con el lenguaje de programación C++ el cual fue explicado en el capítulo **??**. Además, utiliza un lenguaje neutral en la definición de la interfaz (IDL) para describir los mensajes enviados entre los módulos (Figura 5.2).

```
Header header
Point32[] pts
ChannelFloat32[] chan
```

Figura 5.2: *Cabecera IDL.*

C. Herramientas

Con el objetivo de poder gestionar la complejidad de ROS, se ha optado por un diseño *"microkernel"*[1], donde numerosas herramientas se utilizan para generar y ejecutar diferentes operaciones como pueden ser:

- Navegar por el código fuente.
- Obtener y establecer los parámetros de configuración.
- Medir la utilización del ancho de banda.
- Graficar datos de mensajes.

Las ventajas que se consiguen son una mayor estabilidad y una reducción de la complejidad, por contra se pierde eficiencia.

[1] microkernel=micronúcleo del sistema operativo

D. Modular

Debido a la dificultad que tiene la reutilización de código fuera del contexto original para el que fue creado, ROS dispone de librerías que permiten extraer código y reutilizarlo más allá de su intención original, siendo únicamente necesario crear pequeños ejecutables que facilitan la etapa de pruebas.

ROS se caracteriza por reutilizar código de otros proyectos de código abierto como por ejemplo: simuladores del proyecto Player[38], algoritmos de visión de OpenCV [15], algoritmos de planificación de OpenRAVE [7], entre muchos otros. Hay que decir, que el sistema de generación de ROS es capaz de actualizar automáticamente el código fuente de repositorios externos.

En este proyecto los experimentos de simulación se han hecho utilizando código compatible con Player/Stage, obtenidos del repositorio de ROS.

E. Código libre y abierto

El código fuente completo de ROS está disponible al público. Además, se distribuye bajo licencia BSD que permite tanto el desarrollo de proyectos comercializables como no comercializables. ROS pasa los datos entre módulos usando IPC (comunicación entre procesos) y no necesita que dichos módulos estén unidos al mismo ejecutable.

5.3. Conceptos Fundamentales

En esta sección vamos a describir los conceptos fundamentales para poder comprender como funciona la plataforma ROS, toda la documentación necearia está disponible en la página web de la organización [39].

5.3.1. Nodos

Los nodos son los módulos o procesos que llevan a cabo la computación. En otras palabras, no es más que un ejecutable dentro de un paquete de ROS. El nombre de los nodos

debe ser único, usan las librerías-clientes para comunicarse con otros nodos. Pueden publicar o suscribirse a "Topics", así como proporcionar o usar "Service", conceptos que serán explicados más adelante.
Un sistema puede contener varios nodos, por ejemplo:

- Un nodo que controla el dispositivo láser.
- Un nodo que controla los motores de las ruedas.
- Un nodo que lleva a cabo la localización.
- Un nodo que lleva a cabo la planificación.

ROS conecta los nodos de forma dinámica en tiempo de ejecución, las instrucciones que se utilizan a la hora de trabajar con nodos son las siguientes:

- **rosnode** = ros+node : herramienta que nos proporciona información sobre un nodo.
- **rosrun** = ros+run : ejecuta un nodo de un paquete determinado.

Aunque se verá con más detalle en el capítulo seis, sirva de introducción que en el caso de un problema SLAM los nodos necesarios serán: nodo de odometría, nodo de láser, nodo de transformadas y nodo slam.

5.3.2. Mensajes

Los nodos usan mensajes para comunicarse. Un mensaje es un archivo de texto en el que se describen los campos utilizados en el mensaje ROS. Están definidos en el directorio "package-name/msg/*.msg" y son enviados a través de los topics. Los tipos de datos básicos que utiliza son:

- int {8,16,32,64}, float {32,64}, string, time, duration, array.
- **Ejemplo**:
  Point.msg
  float64 x
  float64 y
  float64 z

Entre los comandos utilizados por ROS se encuentran:

- **rosmsg** = ros+msg : proporciona información relacionada con la definición del mensaje ROS.

Este tipo de comunicación es el empleado por los dos algoritmos implementados en este proyecto para enviar los datos de láser.

5.3.3. Topic

Un nodo envía un mensaje publicándolo en un determinado "Topic". Cuando un nodo está interesado en un cierto tipo de dato se suscribirá al topic correspondiente. Puede haber múltiples editores y suscriptores simúltaneos para un mismo topic, y además un mismo nodo puede publicar o suscribirse en diferentes topics. En general, los editores y suscriptores no conocen la existencia de los demás. La comunicación se realiza como muestra la figura 5.3.

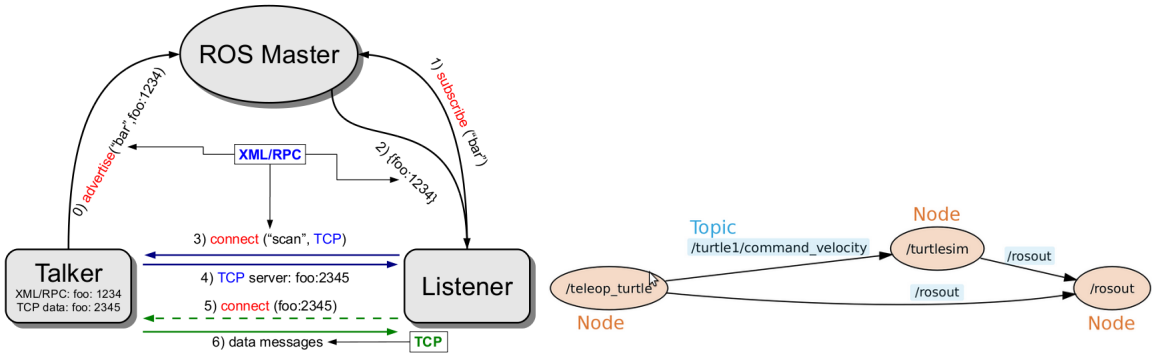


Figura 5.3: *Comunicación ROS-topic.*

En el esquema anterior, podemos ver el nodo "turtlesim-node" y el nodo "turtle-teleop-key" que se comunican entre sí a través del topic "/turtle1/command-velocity".

Las siguientes instrucciones permiten trabajar con este concepto:

- **rostopic** = ros+topic : nos permite obtener información sobre un topic.
- **rxgraph** . crea un gráfico dinámico de lo que está pasando en el sistema. Forma parte del paquete "rxtools" de ROS.

Aunque en el capítulo destinado al desarrollo de los algoritmos se entenderá mejor este concepto, adelantar, que en nuestro caso hemos definido tres topic: base_scan, odom y tf, cada uno de ellos permitirá obtener datos de láser, odometría o llevar a cabo las transformaciones necesarias entre los diferentes sistemas de referencia, respectivamente.

5.3.4. Servicios

Los servicios proporcionan una comunicación tipo pregunta/respuesta. Es decir está formado por dos partes: un mensaje que es el que pregunta y otro mensaje que es el que responde. A diferencia de los topics, un nodo sólo puede llamar a un servicio bajo un nombre particular. La siguiente imagen nos permite ver como se establece dicha comunicación (Figura5.4).

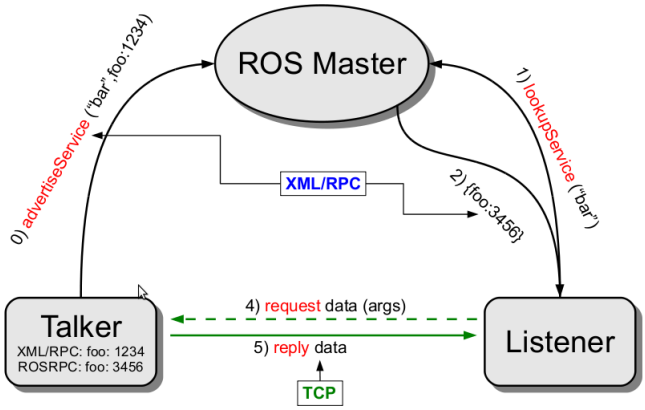


Figura 5.4: *Comunicación ROS-services.*

Los algoritmos que hemos empleado para resolver el problema SLAM utilizan este tipo de comunicación a la hora de obtener los datos del mapa.

5.3.5. Master

Es el nodo principal de ROS, proporciona servicios de registro y búsqueda de nombres dentro del resto del esquema de ejecución. Sin él, el resto de nodos no serían capaces de encontrarse, intercambiar mensajes o solicitar servicios. Se lanza a través de la instruccion ***roscore***. Es la primera instrucción que debemos ejecutar cuando usamos ROS.

5.4. Casos de Uso

A continuación describiremos situaciones comunes que pueden darse cuando se trabaja en el marco del software de la robótica, y como ROS a través de sus recursos es capaz de crear herramientas para responder a las mismas.

5.4.1. *Depuración de un sólo nodo*

El primer paso para poder iniciar una investigación en el ámbito de la robótica debe consistir en definir los límites del sistema en el que vamos a trabajar, de está manera nos podremos centrar por ejemplo en un nodo que realiza localización, o un nodo que se encarga de la percepción o del control. Sin embargo, para poner en funcionamiento el sistema robótico es necesario que exista un ecosistema de software que una de manera adecuada cada parte individual del sistema.

Los nodos están conectados de forma dinámica en tiempo de ejecución, pero sólo se reinician periódicamente aquellos nodos que necesitan modificar su código fuente, siendo el propio sistema quien se encarga de la modificación. Es por esto que ROS está diseñado para minimizar la depuración en casos en los que nuestra aréa de estudio forme parte de un gran sistema, de esta forma se consigue un aumento de la productividad y un sistema más complejo e interconectado.

5.4.2. *Registro y Reproducción*

Cualquier flujo de mensajes de ROS puede ser almacenado en el disco y reproducirse más tarde. Todo esto puede hacerse en la línea de comandos y no requiere la modificación del código fuente de ninguna de las piezas software. Para facilitar el registro y seguimiento de los sistemas distribuidos a través de muchas máquinas, la librería ***rosconsole*** se basa en el sistema log4cxx del proyecto Apache y proporciona una interfaz en la cual todos los registros se publican en el topic ***rosout***.

5.4.3. *Subsistemas*

La investigación robótica suele desarrollar aplicaciones de gran tamaño en la que es necesario la existencia de varios nodos interconectados, cada uno de los cuales tienen muchos parámetros. Un buen ejemplo donde se da esta situación es en la navegación de robot, mostramos un gráfico (Figura 5.5) en el que se ven los nodos necesarios y la dependencia entre ellos.

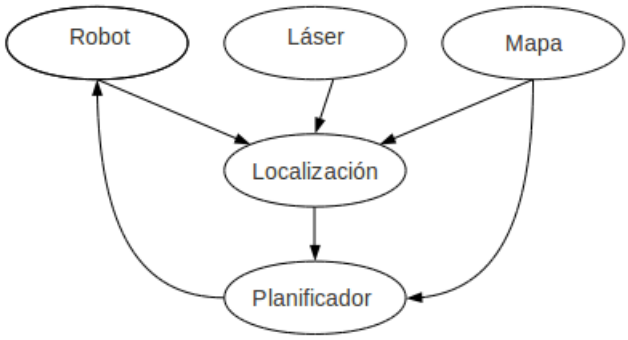


Figura 5.5: *Sistema de navegación.*

Cada uno de los nodos anteriores puede ser ejecutado en una línea de comandos con su instrucción correspondiente, pero esta labor puede ser muy tediosa. Para solventar este problema, existe una herramienta en ROS llamada ***roslaunch*** que permite de forma fácil poner en marcha múltiples nodos de ROS a nivel local y remoto a través de SSH, así como los parámetros de configuración en el servidor de parámetros.

Los *roslaunch* están definidos como archivos de configuración XML (con la extensión de su lanzamiento), en los cuales se especifican los parámetros necesarios para establecer y poner en marcha los nodos, además de indicar las máquinas en que deben ejecutarse. Esta funcionalidad ayuda de manera significativa no sólo al intercambio y a la reutilización, sino también a la integración, instalación y desmontaje de grandes sistemas distribuidos.

Esta herramienta nos ha sido de gran utilidad y nos ha simplificado el trabajo ya que en el problema SLAM intervienen multitud de nodos, cada uno de los cuales necesita ser ejecutado en un terminal distinto. Creándonos un archivo .launch hemos conseguido ejecutar todos ellos en un único terminal y al mismo tiempo poner en funcionamiento al nodo maestro que a su vez comienza a comunicar las ordenes necesarias a todos los nodos implicados. En el apéndice C se puede consultar un ejemplo de como ha sido programado.

5.4.4. *Proyectos de colaboración.*

La construcción de grandes sistemas robóticos requiere de la colaboración entre investigadores, con el fin de apoyar este desarrollo colaborativo el sistema de software de ROS se organiza en paquetes. Un paquete en ROS no es más que un directorio que contiene un archivo XML en el que encontramos una descripción del mismo y de sus dependencias. El sistema de archivos se organiza de la siguiente manera:

a) **Packages**[2] Son la unidad principal en la que se organiza el software en ROS. Un paquete puede contener procesos ejecutables (nodos), librerías dependientes de ROS, archivos de configuración etc.

b) **Manifest**[3] Proporcionan información sobre un paquete, incluyendo tipo de licencia de uso y publicación, sus dependencias de otros paquetes e información específica de compilación.

c) **Stacks**[4] Son conjuntos de paquetes que proporcionan una funcionalidad concreta, es la manera en la que el software es publicado contando con números de versión asociados. Al igual que los paquetes llevan asociado un manifesto.

ROS proporciona dos instrucciones para trabajar con estas herramientas:

- **rospack** = ros+pack: herramienta que nos permite encontrar los paquetes por su nombre, consultarlos, inspeccionarlos y ver sus dependencias.
- **rosstack** = ros+stack

[2]Package = paquete
[3]Manifest = manifiesto
[4]Stack = pila

En el siguiente esquema (Figura 5.6) vemos un desglose de como organiza ROS el nivel del sistema de archivos.

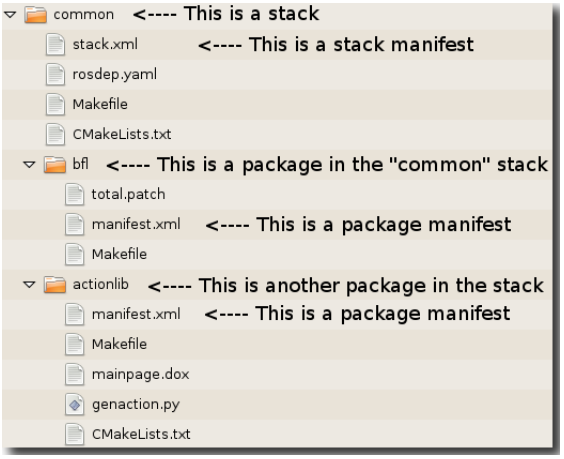


Figura 5.6: *Sistema de archivos.*

En nuestro proyecto hemos creado dos paquetes, uno para cada uno de los algoritmos implementados: coreslam y slam_gmapping. En ellos se almacenan todos los recursos necesarios, tales como carpetas con los diferentes archivos .launch, carpetas en las que se van almacenando los .bag[5] que vamos grabando, etc.

5.4.5. *Visualización y Monitorización*

Durante la etapa de diseño y depuración del software se hace necesario observar distintos estados mientras el sistema se está ejecutando. ROS es capaz de acceder al sistema gráfico y observar estos estados ya que desarrollado un programa visualizador para tal fin llamado ***rviz*** (Figura 5.7).

[5]aunque se definira más adelante, decir que los ficheros .bag nos permitirán almacenar datos de odometría y láser.

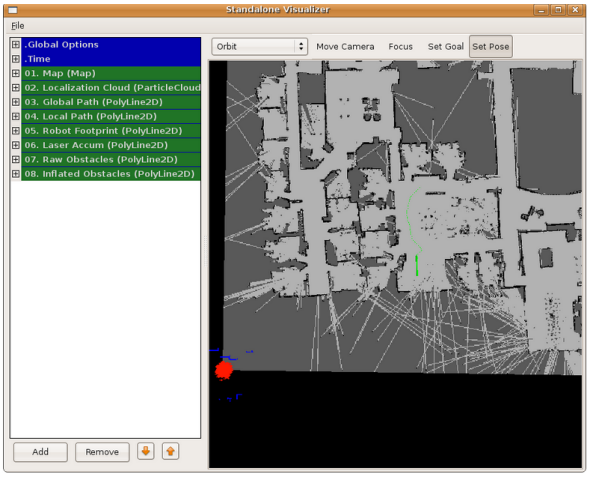


Figura 5.7: *Rviz.Herramienta de visualización desarrollada por ROS.*

La herramienta rviz nos ha sido de gran utilidad ya que permite visualizar el funcionamiento del algoritmo, además resulta muy atractiva para mostrar a las personas que no están familiarizadas con el campo de la robótica como se va modelando un mapa del entorno. Aunque tiene la ventaja de ser muy potente, a nosotros nos ha supuesto una limitación pues el computador portátil en el que comenzamos a desarrollar el proyecto no tenía la potencia necesaria para soportar esta herramienta siendo necesario recurrir a un computador de mesa con las caracteriticas adecuadas.

Existen otras herramientas de línea de comandos que nos permiten visualizar un gráfico de cómputo de ROS, es el caso de ***rxgraph*** que nos muestra los nodos que se están ejecutando (representados por óvalos), así como los topic que los conectan (arcos). La siguiente figura 5.8 nos muestra el resultado de aplicar esta instrucción al algoritmo Gmapping en el caso de trabajar con el manipulador autónomo antropomórfico Manfred.

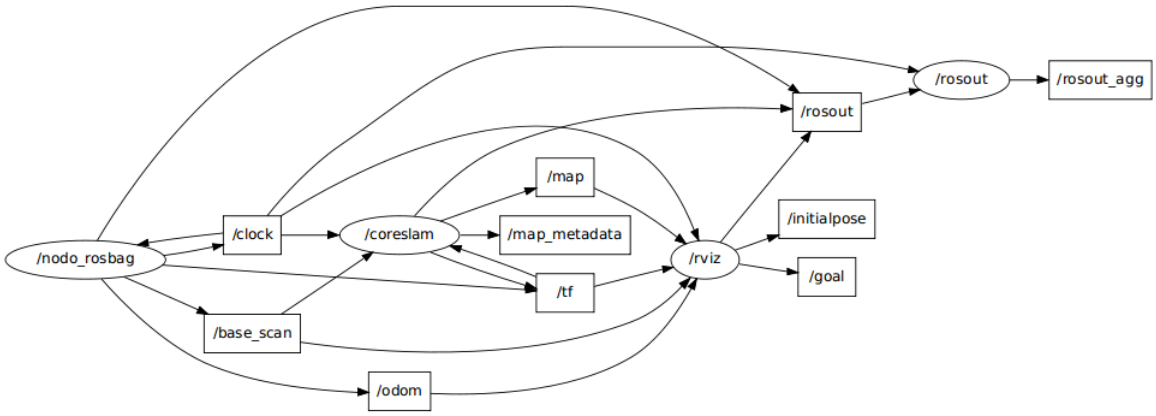


Figura 5.8: *Resultado de ejecutar la instrucción rxgraph de ROS.*

5.4.6. *Transformaciones*

Los sistemas robóticos necesitan de la existencia de relaciones espaciales, por ejemplo: entre un robot móvil y un marco de referencia fijo para la localización, entre varios marcos de diferentes sensores, etc. Para simplificar y unificar el tratamiento de los marcos espaciales, ROS ha escrito un sistema de transformación llamado *tf*. El cual construye un árbol de transformación dinámica que relaciona todos los marcos de referencia del sistema.

En el caso del problema SLAM se necesita un árbol de transformadas entre el láser, la base del robot, la odometría y el mapa. En el capítulo seis mostramos el diagrama de transformadas que se obtiene al ejecutar el algoritmo Gmapping en el robot real.

5.4.7. *Grabación de datos*

Un *bag*, es un formato de archivo de ROS que permite almacenar datos que podrán ser utilizados posteriormente por algoritmos de prueba o de visualización. Los *bags* son creados por la herramienta ***rosbag***, se suscriben a uno o más *topics* y almacenan en un archivo los datos del mensaje recibido. Estos *bags* podrán reproducirse en ROS a partir del *topic* en el que fueron grabados, incluso podrán ser reasignados a otros *topics* distintos.

En nuestro caso hemos utilizado los archivos .bag para grabar los datos de láser, odometría y transformación necesarios para ejecutar el algoritmo SLAM. Nos han sido de gran ayuda ya que para poder comparar el funcionamiento de los algoritmos implemetados (CoreSlam y Gmapping), era imprescindible que los datos enviados a ambos procesos fueran los mismos.

5.5. Conclusión

ROS es un experimento de ingeniería de software que permite la facilidad de desarrollo, pudiendo los investigadores centrarse en su área específica.
Su diseño abierto permite que pueda ser ampliado y aprovechado para construir diferentes sistemas softwares para robots, útiles para una gran variedad de plataformas hardware, investigación y requerimientos de tiempo de ejecución.

Capítulo 6

Desarrollo

En este capítulo desarrollaremos y explicaremos los algoritmos con los que hemos trabajado: “CoreSlam” y “Gmapping”. El funcionamiento de ambos, ha sido probado tanto con simulación, usando la herramienta Player/Stage, como con robots reales, trabajando con el robot móvil autónomo antropomórfico Manfred.

En primer lugar se explicarán las diferencias, ventajas y desventajas que puede suponer el trabajar con simulación en comparación con trabajar en entornos reales. A continuación, nos centraremos en las bases conceptuales funcionamiento de los algoritmos empleados.

6.1. Simulación VS Realidad

Los robots son piezas complejas de Hardware compuestas por múltiples mecanismos y dispositivos, los cuales en su conjunto, hacen posible que dichas máquinas puedan llegar a realizar acciones e incluso simular comportamientos que en algunos casos pueden considerarse autónomos. Si bien es cierto que la parte mecánica de los robots es fundamental, sin el correspondiente Software de control, los robots no serían capaces de conseguir cierta inteligencia y autonomía

La programación permite controlar las distintas piezas de las que se compone un robot, de forma que estas máquinas puedan realizar multitud de tareas diferentes. Para programar

el control de un robot, siempre ha sido necesario utilizar métodos y lenguajes específicos muy ligados al hardware del robot para el que se pretendía programar, lo que implicaba muy poca flexibilidad y por tanto hacía imposible la estandarización de los aspectos de la programación de robots.

Hoy en día la situación está cambiando, gracias a la aparición de plataformas y herramientas que generalizan la programación de los robots y reducen la dependencia de los programas de control con el hardware específico. Actualmente, el interés por los robots ha aumentado considerablemente, gracias a las nuevas posibilidades que ofrecen los avances tecnológicos, que permiten dotar a estas máquinas de una mayor capacidad de procesamiento y almacenamiento, así como de sensores mucho más preciso.

De los pequeños programas para robots, los cuales implementaban tareas muy sencillas, se ha pasado a programar tareas muchos más complejas cuyos requerimientos y sistemas son mayores y más avanzados. Un ejemplo lo podemos encontrar en los computadores personales y en las herramientas de alto nivel que ofrecen la ventaja de la abstracción o reutilización de código, aunque por contra pierden eficiencia. Aunque la tecnología no suponga un problema, si que se encuentran limitaciones debido al alto desembolso que supone adquirir un robot. Éste es uno de los motivos por los cuales se utilizan simuladores para probar los programas y comprobar los resultados obtenidos, acercándose lo más posible a los que podrían obtenerse con el robot real.

El uso de un simulador permite ejecutar los programas como si del propio robot se tratase, aunque tenemos que tener en cuenta que las condiciones del entorno simulado no son las mismas que las del entorno real, el cual presenta ruidos en las mediciones de los sensores o situaciones inesperadas, situaciones que a veces no pueden ser reproducidas con fidelidad en la simulación. Los simuladores suelen ser utilizados para comprobar los programas que posteriormente serán implementados en el robot real de una forma más metódica y exhaustiva, ahorrando en algunos casos bastante tiempo.

Para llevar a cabo nuestro proyecto, primeramente se han probado los algortimos CoreSlam y Gmapping en la herramienta de simulación Player/Stage (consultar capítulo 4), y más tarde se ha procedido a trabajar en la plataforma Manfred.

Trabajar con simulación conlleva una serie de ventajas y desventajas:

- La simulación permite estudiar el efecto de cambios internos y externos del sistema, al realizar modificacones en su modelo y observar los efectos de esas alteraciones en el comportamiento del mismo.
- La simulación ayuda a entender mejor el sistema, y por consiguiente es posible sugerir estrategias que mejoren la operación y eficiencia del mismo.
- Las técnicas de simulación puede ser utilizadas para experimentar con nuevas situaciones sobre las cuales se tiene poca o ninguna información. A partir de esta experimentación se puede anticipar y mejorar posibles resultados no previstos.
- Los modelos simulados permiten reducir el tiempo de experimentación. El equivalente de días, semanas y meses de un sistema real pueden ser simulados en solo segundos, minutos u horas en una computadora, lo que permite simular mayor número de soluciones obteniendo los resultados de forma breve pudiendo elegir el diseño más adecuado para el sistema.

En cuanto a las desventajas que presentan las técnicas de simulación podemos citar:

- La simulación sólo nos ofrece resultados estimados, fallando a la hora de producir resultados exactos.
- La simulación no es una técnica de optimización, es decir, es usada para contestar preguntas del tipo ¿qué pasa si?, pero no de, ¿qué es lo mejor?. La simulación no genera soluciones, sólo evalúa las que han sido propuestas.

6.2. Implementación

En capítulos anteriores ya hemos explicado que la Localización y Modelado Simultáneo (SLAM) consiste en adquirir un mapa del entorno mientras que simultáneamente el robot busca localizarse en relación a dicho mapa. Por otro lado, hemos estudiado los diferentes algoritmos que se han propuesto para tratar de resolver el problema de localización y slam en robótica móvil. El siguiente paso consistirá en trabajar con los algoritmos CoreSlam y Gmapping, estudiando como de buenas son las soluciones que ofrecen al problema SLAM.

Como comentamos al comienzo de esta sección, hemos trabajado tanto con datos simulados como con datos reales. En el caso de la simulación utilizamos el paquete "stageros", disponible en ROS, que nos permite obtener y tratar datos procedentes de un dispositivo láser y medidas de posición del robot. Los datos reales son proporcionados por el robot Manfred. En este caso, para poder emplear la herramienta ROS hemos tenido que desarrollar todos los nodos implicados en el problema SLAM, esta labor se ha hecho en colaboración con el resto del Departamento de Automática que trabaja en el proyecto Manfred.

6.2.1. CoreSlam

El paquete CoreSlam pertenece al proyecto OpenSlam, encontrándose disponible en el repositorio de ROS. Los autores del proyecto son Bruno Steux y Oussama El Hamzaoui, mientras que el repositorio ROS es obra de Michael Ferguson.

Como acabamos de decir, Coreslam utiliza uno de los algoritmos pertenecientes al proyecto OpenSLam, más concretamente hace uso del algoritmo SLAM llamado tinySLAM. Debido a la sencillez del algoritmo, está escrito en lenguaje C y programado en 200 líneas de código, decidimos iniciar nuestro trabajo basándonos en esta implementación del problema de Localización y Modelado Simultáneo. El proyecto tinySLam se desarrolla en el año 2010, la idea era desarrollar e implementar un algoritmo de SLAM muy simple que pudiese ser integrado en un sistema de localización basado en filtro de partículas.

Más adelante desarrollaremos el funcionamiento del algoritmo, pero vease un mapa (Figura 6.1) obtenido utilizando esta técnica.

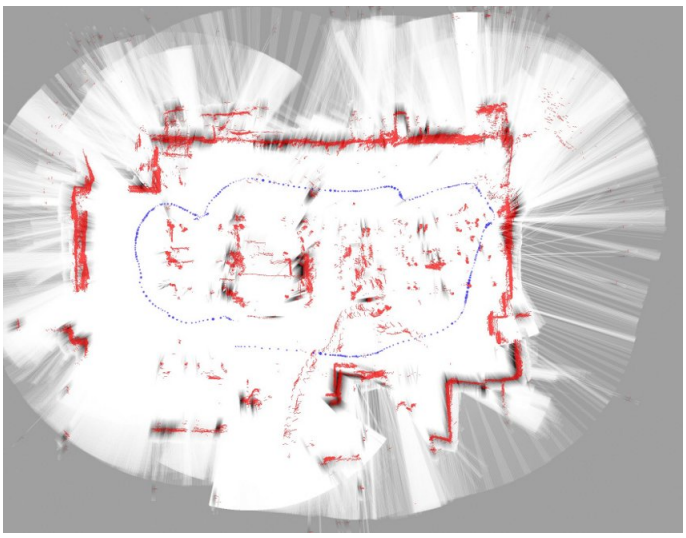


Figura 6.1: *Mapa del laboratorio CAOR. Como datos de entrada recibe valores brutos de láser y odometría.*

Para poder hacer uso del paquete CoreSlam de ROS, el primer paso consistió en descargarse del servidor el código. Una vez hecho ésto, lo único que necesitábamos era un robot móvil que proporcionase datos de odometría y medidas de un dispositivo láser; en nuestro caso los datos fueron simulados utilizando Player/Stage por un lado, y por otro se obtuvieron medidas a través de Manfred. El nodo slam_coreslam fue el encargado de transformar cada ciclo de entrada de datos de odometría a su correspondiente marco *tf*.[1] Realizadas todas estas operaciones estábamos en disposición de probar el funcionamiento del algoritmo, para lo cual terníamos que ejecutar la siguiente instrucción:

```
$ rosrun coreslam slam_coreslam scan:=base_scan
```

El nodo slam_coreslam toma el mensaje del tipo sensor_msgs/laserScan y construye un mapa como mensaje tipo nav_msgs/OccupancyGrid. El mapa se recupera via ROS[2] mediante el topic o service correspondiente.

[1] Como se explicó en el capítulo dedicado a ROS, tf es el sistema de transformación integrado en ROS que permite establecer las relaciones entre los diferentes marcos de coordenadas que nos encontramos en un robot móvil.

[2] Todos los conceptos de la plataforma de programación ROS fueron estudiados en el capítulo 5

Algoritmo CoreSlam

CoreSlam fue desarrolado con el objetivo de ser un algoritmo simple, fácil de entender y que ofreciese un buen rendimiento. Lo que le diferencia de otros algoritmos basados en el uso de filtros de partículas es que CoreSlam sólo usa un mapa, en lugar de asociar un mapa a cada partícula. Para incorporar la infomación láser al subsistema de localización haciendo uso de un filtro de partículas se emplean dos funciones:

1) La función de distancia al mapa tras una exploración que actúa como función de probabilidad comprobando la hipótesis del estado de posición (partícula) en el filtro. La función es llamada ts_distance_scan_to_map y consiste en una suma de todos los valores del mapa a todos los puntos de impacto de la exploración (en relación a la posición de la partícula).

2) La función de actualización del mapa ts_map_update que es utilizada para construir el mapa a medida que el robot avanza.

Para construir el mapa es necesario controlar los picos de la función de verosimilitud, para ello CoreSlam hace uso de mapas de niveles de gris. La actualización del mapa consiste en la creación de agujeros cuya anchura se corresponde con los picos de dicha función de verosimilitud.

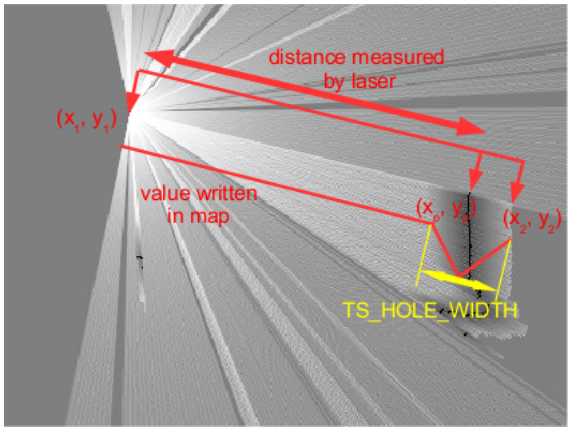


Figura 6.2: *Resultado de la integración en el mapa de un escaneo láser. Se observa la "V" del agujero dibujado alrededor de cada impacto de láser.*

Para cada obstáculo detectado, el algoritmo no saca un sólo punto, sino una función con un agujero cuyo punto más bajo está en la posición del obstáculo (Figura 6.2). La integración en el mapa se realiza a través de un filtro resultando una convergencia del mapa a los perfiles más nuevos. La función de actualización del mapa es llamada desde la última posición actualizada del filtro de partículas, siendo ésta el promedio ponderado de todas las partículas después de la evaluación de la probabilidad.

La función ts_map_update utiliza la función ts_map_laser_ray, que a su vez usa un algoritmo *Bresenham*[3] para dibujar los rayos láser en el mapa. Es muy importante que este procedimiento sea rápido ya que gran parte del marco del mapa es rastreado por un escáner láser. La función del agujero es actuar como guía y hacer que el algoritmo de coincidencia converga con mayor facilidad al obstáculo. CoreSlam utiliza un algoritmo de Monte-Carlo para hacer coincidir el ciclo actual con el mapa y para recuperar la posición actual del robot.

Existen dos cuestiones relativas a la precisión del mapa y al tiempo de latencia de integración de los datos en el mapa, que deben ser explicadas.

a) Aunque la resolución del mapa sea de 1cm es posible medir desplazamientos menores, ya que la función ts_distance_scan_to_map tiene en cuenta varios puntos para hacer sus cálculos.

b) En cuanto al tiempo entre el escaneo láser y su integración en el mapa, es necesario medir desplazamientos pequeños en relación con la resolución del mapa.

[3]El algoritmo Bresenham, es considerado uno de los algoritmos más efectivos para dibujar líneas mediante rastreo. Emplea cálculos incrementales con valores enteros.[4] [5]

6.2.2. Slam_Gmapping

El paquete gmapping también pertenece al proyecto OpenSlam, que se encuentra en los repositorios de ROS. Los autores del código son Giorgio Grisetti, Cyrill Stachniss, Wolfram Burgard, el repositorio de ROS es obra de Brian Gerkey.

Este método utiliza un filtro de partículas (consultar capítulo 3) en el que cada partícula lleva un mapa individual del entorno, lo que supone un elevado número de partículas que habrá que intentar reducir. Para resolver este problema, Gmapping presenta una técnica de adaptación por la cual, mediante un filtro de partículas Rao-Blackwellize [4] se logra una disminución de la cantidad de muestras. Lo que propone, es un enfoque para calcular la distribución exacta propuesta teniendo en cuenta no sólo el movimiento del robot sino también la observación más reciente. Esta drástica disminución de la incertidumbre sobre la pose del robot se lleva a cabo en el paso de predicción del filtro. Además, se aplica un enfoque selectivo para llevar a cabo la operación de remuestreo que reduce notablemente el problema de la reducción de partículas.

Mostramos un mapa (Figura6.3), resultado de aplicar el algoritmo Gmapping.

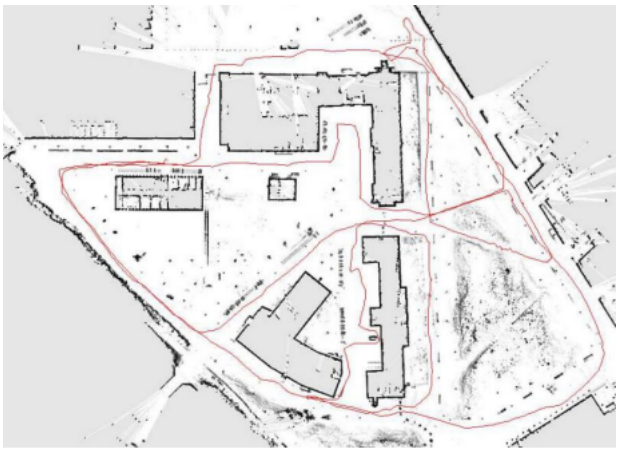


Figura 6.3: *Mapa generado por SLAM (Grisetti, Stachniss e Burgard 2006)*

[4]Recientemente los filtros de partículas Rao-Blackwellize se han ido introduciendo como solución al problema de Localización y Modelado Simultáneo (SLAM).

Al igual que ocurría con el algoritmo CoreSlam, para usar slam_gmapping necesitamos un robot móvil que proporcione datos de odometría y que además esté equipado con un telémetro láser. De nuevo, obtendremos estos datos mediante simulación y a través de las medidas tomadas por Manfred. El nodo slam_gmapping transformará cada ciclo de entrada de datos de odometría al marco adecuado. En esta ocasión la instrucción que ejecuta el paquete es:

```
$ rosrun gmapping slam_gmapping scan:=base_scan
```

El nodo slam_gmapping recibe un mensaje tipo sensor_msgs/laserScan y construye un mapa como mensaje tipo nav_msgs/OccupancyGrid. El mapa se recupera via ROS mediante el topic o service correspondiente. Debemos comprobar el nombre del nodo que está enviando por el láser, ya que en ROS siempre tenemos que verificar que el nodo suscriptor y publicador tienen el mismo nombre para el topic con el que se están comunicando.

6.2.2.1. Algoritmo Gmapping

Murphy, Doucet junto con otros compañeros, introdujeron el Filtro de Partículas Rao-Blackwellize como medio eficaz para resolver el problema de Localización y Modelado Simultáneo. La complejidad de este método reside en el número de partículas necesarias para construir un mapa exacto, la reducción de esta cantidad de partículas es uno de los mayores desafíos para esta familia de algoritmos. A lo que hay que unir, que durante la etapa de remuestreo pueda eliminarse la partícula correcta, éste efecto es conocido como *El problema del agotamiento de la partícula* o *Empobrecimiento de la partícula.*

Existen diferentes métodos que permiten aumentar el rendimiento de filtro de partículas Rao-Blackwellize cuando es aplicado en la solución del problema SLAM en un mapa de celdillas:

a) Una distribución propuesta que considera la exactitud de los sensores del robot y nos permite extraer partículas de manera precisa.

b) Un remuestreo que mantiene una variedad razonable de partículas y que permite que el algoritmo obtenga un mapa preciso, mientras que reduce el riesgo de agotamiento de partícula.

La distribución propuesta se calcula mediante la probabilidad alrededor de la partícula que más depende de la pose obtenida por el procedimiento de *scan-matching* junto con la información de odometría. De esta manera, se tiene en cuenta la observación más reciente del sensor para crear la próxima generación de partículas permitiéndonos obtener el estado del robot de acuerdo a más información que la obtenida en un modelo en el que sólo se tiene en cuenta la odometría.

El uso de este modelo tiene dos efectos:

a) Mapa más preciso ya que la observación actual está incorporada en los mapas individuales de después, lo cual reduce significativamente el error de estimación, de modo que menos partículas son necesarias para representar la parte posterior.

b) El remuestreo sólo se realiza cuando es necesario, manteniendo así una diversidad de partículas razonables que reduce el riesgo de empobrecimiento de partícula.

Podríamos resumir la metodología de trabajo de este algoritmo en los siguientes pasos:

- **Muestreo** La próxima generación de partículas se obtiene de la generación anterior por muestreo de la distribución propuesta.
- **Importancia de los pesos** Se asigna a cada partícula un peso de acuerdo a su importancia según el principio de muestreo.
- **Remuestreo** Algunas partículas se reemplazan proporcionalmente a la importancia de sus pesos, esta etapa sólo es necesaria cuando un número finito de partículas se usa para aproximarse a una distribución contínua. El remuestreo nos permite utilizar un filtro de partículas cuando la distribución se aleja de la propuesta. Después del remuestreo todas las partículas tienen el mismo peso.
- **Estimación del mapa** Para cada partícula, el mapa estimado correspondiente se obtiene basándose en la trayectoria que ha seguido la partícula y en la historia de sus observaciones.

6.3. Implementación Manfred

Para llevar a cabo la implementación de SLAM en ROS y poder trabajar con los datos que nos proporciona Manfred hemos tenido que diseñar cada uno de los nodos que permiten resolver un problema de Localización y Modelado Simultáneo. Cada uno de estos nodos se detallan a continuación, hemos creído conveniente indicar el tutorial de ROS que nos ha servido de guía para llevar a cabo la programación de cada uno de ellos.[5]

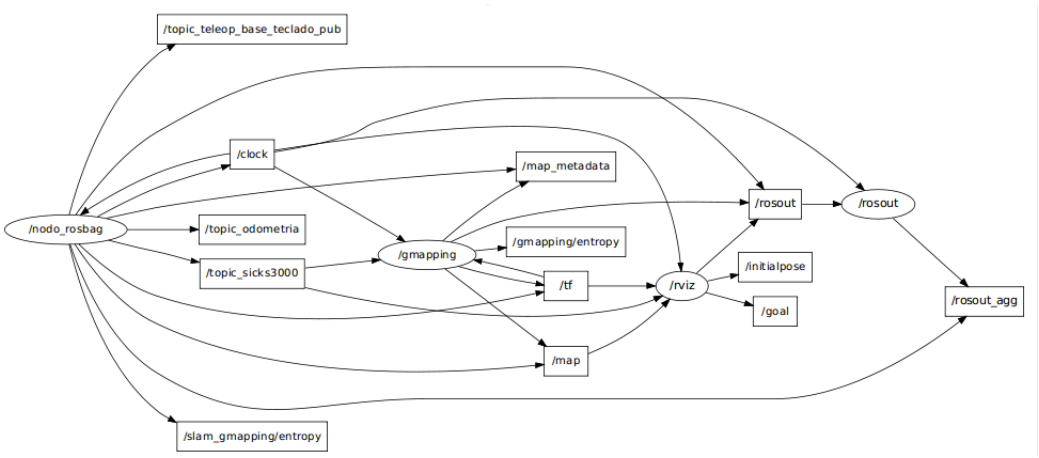


Figura 6.4: *Relaciones y comunicaciones que se establecen entre los distintos nodos mientras se ejecuta el algoritmo Gmapping.*

a) Nodo de odometría: nos permite obtener los datos de la pose del robot.
`http://www.ros.org/wiki/navigation/Tutorials/RobotSetup/Odom`

b) Nodo de láser: necesario para recibir las medidas proporcionadas por el sensor láser.
`http://www.ros.org/wiki/navigation/Tutorials/RobotSetup/Sensors`

c) Nodo de teleoperación: gracias al cual podemos conducir y guiar al robot alrededor del entorno.

d) Rviz. Es la herramienta de visualización utilizada por ROS.
`http://www.ros.org/wiki/rviz/UserGuide`

[5]Recomendamos visitar diferentes *blogs*: http://roslauragalindez.blogspot.com/ , http://www.mrpt.org/ de gente anónima que colabora en Universidades de todo el mundo y que también se dedican al estudio del problema SLAM, ya que ha nosotros nos han sido de gran ayuda a lo largo del proyecto. De esta manera el lector podrá seguir fácilmente la explicación de cómo generar un módulo gmapping.

e) Tf. Necesitamos un árbol de transformadas para que el marco de referencia del láser se traduzca al de la base y luego al de odometría (y posteriormente al del mapa pero eso lo hace el nodo de slam_gmapping o core_slam). Tenemos que construir el siguiente árbol de transformadas:

$base\_scan \rightarrow base\_link \rightarrow odom \rightarrow map$.

En la figura 6.5 vemos el flujo de transformadas que obtenemos al realizar los experimentos con Manfred.

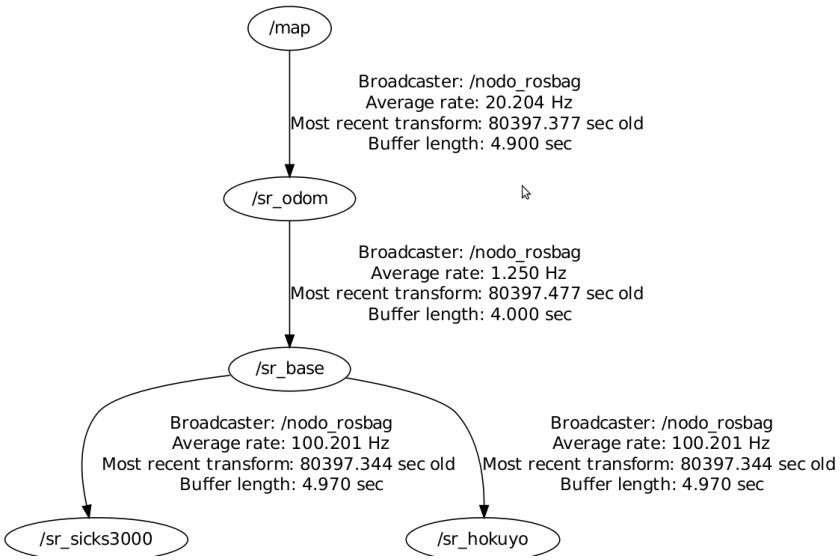


Figura 6.5: *Árbol de transformadas generado en Manfred.*

f) Nodo slam_gmapping. Permite ir construyendo un mapa mientras el robot lleva a cabo una exploración del entorno.

Capítulo 7

Resultados experimentales.

En el capítulo anterior se explicó el funcionamiento de cada uno de los algoritmos utilizados en el proyecto. Una vez comprendida la parte teórica, el siguiente paso consistirá en realizar diferentes experimentos para comprobar su correcto funcionamiento. Para ello, se han realizado varias pruebas con diferentes conjuntos de datos que contienen medidas de odometría y láser, procedentes tanto de entornos simulados como de entornos reales.

7.1. Experimentos y resultados con simulación.

A continuación presentamos los resultados obtenidos tras trabajar con datos simulados. Los mapas han sido construidos aplicando tanto el algoritmo CoreSlam como el algoritmo Gmapping. Los entornos donde el robot ha llevado a cabo la exploración han sido simulados con la herramienta Player/Stage (consultar capítulo 4). Los datos de odometría han sido simulados con un 5 % de ruido. El láser ha sido simulado con un rango máximo de 15 metros y con un campo de visión de 180 grados, el número de medidas obtenido en cada barrido láser es de 180.

Como se explicó en capítulos anteriores, utilizamos datos simulados porque nos permitirán simular los mismos procesos y las mismas condiciones de trabajo tanto en el algoritmo CoreSlam como en el algoritmo Gmapping. De esta forma, podremos comparar el funcionamiento y los resultados que nos proporcionan cada uno de los algoritmos. La

simulación nos permitirá estudiar el comportamiento de los algoritmos empleados antes de ser implementados en Manfred, dándonos la oportunidad de corregir o visualizar cualquier posible anomalía del sistema.

A continuación mostramos los resultados obtenidos en cada uno de los experimentos llevados a cabo. La manera en la que han sido organizados es la siguiente: primero se muestra el entorno original y seguidamente a la izquierda se representa el mapa generado por el algoritmo CoreSlam; a la derecha podemos observar el resultado de aplicar el algoritmo Gmapping.

EXPERIMENTO 1: LABORATORIO

El entorno de este primer experimento (Figura. 7.1), pertenece al repositorio Player. Se trata de un entorno con varias habitaciones, a través de las cuales hemos ido teleoperando al robot para que llevase a cabo una exploración de la misma. Los resultados obtenidos son los que se muestran en la figura 7.2.

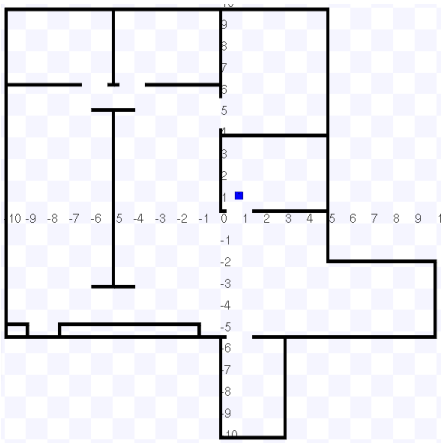
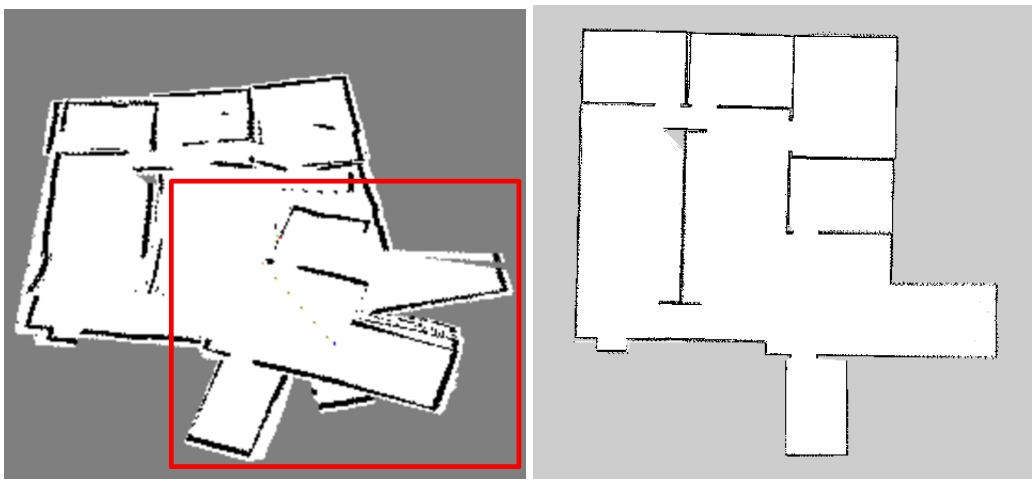


Figura 7.1: *Entorno Original*



(a) Resultado CoreSlam (b) Resultado Gmapping

Figura 7.2: *Laboratorio.*

Observamos que los mejores resultados se obtienen con el algoritmo Gmapping, en el caso de CoreSlam vemos que se produce un error en la esquina inferior derecha. Se produce ya que el láser recibe poca información debido a la amplitud de la zona explorada, donde la mayoría de los obstáculos están más alejados que el rango máximo del sensor (ajustado, recordemos, a 15 metros). Fundamentalmente, este fallo se debe a la incapacidad del algoritmo de cerrar el bucle en este punto, comienzo y final de la trayectoria simulada, debido al error acumulado durante la construcción del mapa. Se puede observar cómo el algoritmo Gmapping es mucho más preciso, al conservar el paralelismo entre las paredes modeladas, a lo largo de todo el experimento.

Por otro lado, los resultados obtenidos por CoreSlam son peores ya que el algoritmo sólo realiza un mapa, mientras que Gmapping genera 30 mapas sobre los que filtra, quedándose con el más probable.

EXPERIMENTO 2: HABITACIÓN

En el segundo experimento llevado a cabo, se condujo al robot a lo largo del pasillo y se inspeccionaron cada una de las habitaciones que se encuentran en los laterales.

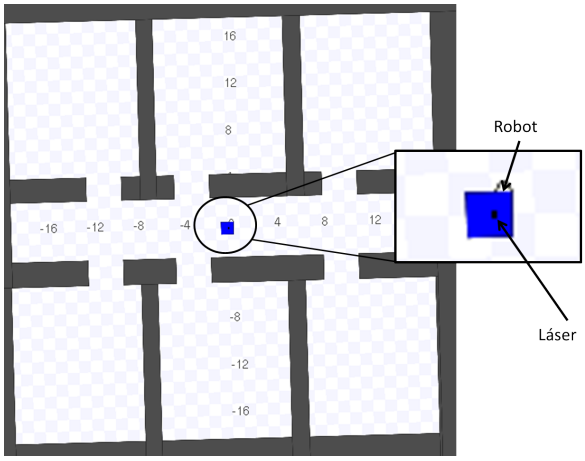
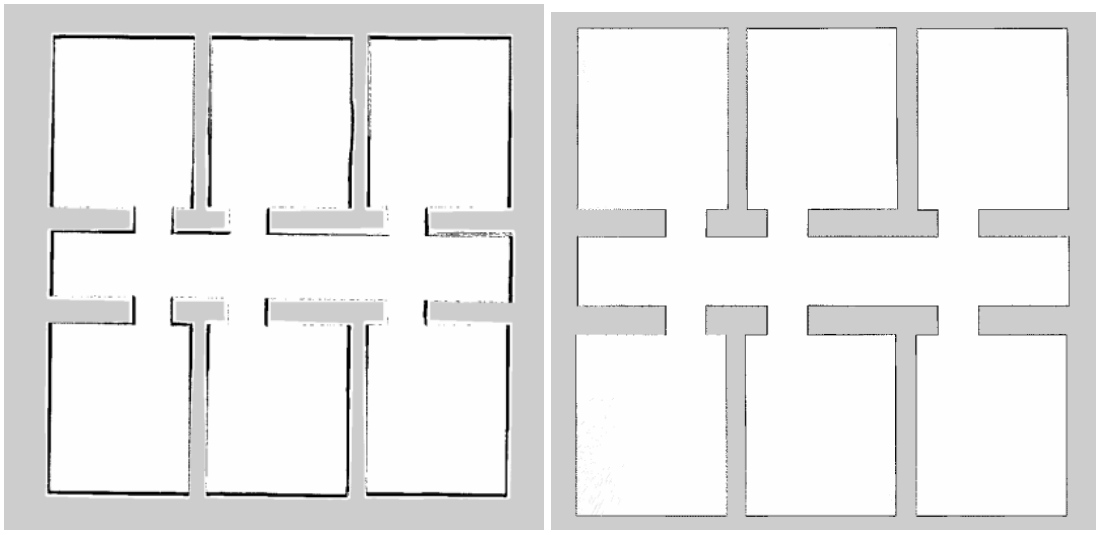


Figura 7.3: *Entorno Original*



(a) Resultado CoreSlam (b) Resultado Gmapping

Figura 7.4: *Habitación.*

Los dos mapas construidos en este caso por ambos algoritmos son de calidad comparable, obteniendo una representación bastante fiel del entorno original. Esto se debe, a que en este caso los espacios visitados son menos amplios y la cantidad de información adquirida por el sensor láser en cada muestra es mayor.

EXPERIMENTO 3: LABERINTO

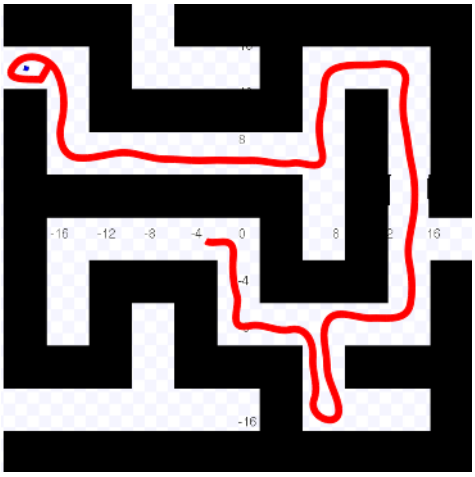
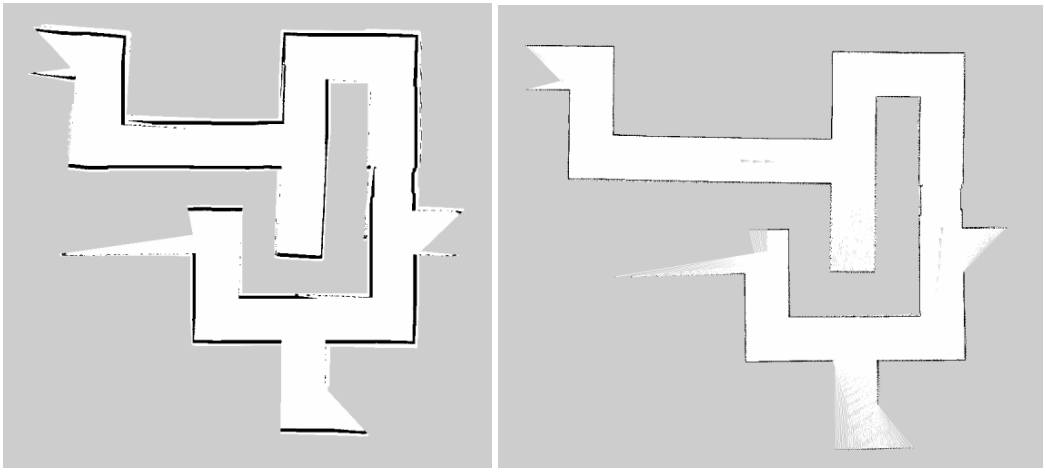


Figura 7.5: *Entorno Original*



(a) Resultado CoreSlam (b) Resultado Gmapping

Figura 7.6: *Semi-Laberinto.*

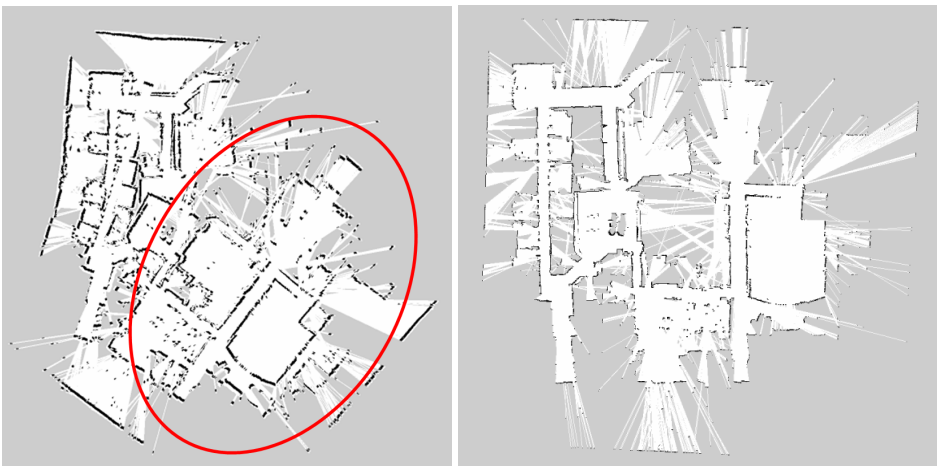
Para la tercera prueba escogimos un entorno en el que la complejidad era mayor. Como observamos en la figura 7.5, el entorno está fomado por una serie de pasillos estrechos y quiebros que hacen que el robot deba desplazarse a menor velocidad, evitando dar giros bruscos. La trayectoria que realiza el robot es la señalada en color rojo.

Cuando se trabaja en entornos como éste, donde existen pasillos, la información que recibe el sensor láser varía en función de si nos referimos a información longitudinal, que suele ser poca, o si nos referimos a información lateral, que es mayor, y que por tanto, permite corregir las posibles desviaciones que se produzcan en esta dirección.

EXPERIMENTO 4: LABORATORIO WILLOW

El último de los experimentos realizados en este bloque, al igual que en los casos anteriores, fue generado con datos simulados pero obtenidos a partir de un mapa generado a su vez con datos procedentes de un entorno real (los laboratorios de la empresa Willow Garage, responsable de ROS, y disponible en la distribución).

Figura 7.7: *Entorno Original*



(a) Resultado CoreSlam (b) Resultado Gmapping

Figura 7.8: *Laboratorio Willow.*

En el caso de CoreSlam, puede observarse que se trata de un algoritmo mucho menos robusto. Cuando el algoritmo dispone de poca información sensorial, ó cuando el algoritmo de localización falla, no es capaz de recuperarse del error cometido. Esto se debe, como explicamos en el capítulo 6, a que el algoritmo CoreSlam a pesar de estar basado en un filtro de partículas, sólo usa un mapa en lugar de asociar a cada partícula un mapa.

Los resultados obtenidos por el algoritmo Gmapping son bastante buenos y se asemejan al entorno original. Este método utiliza un filtro de partículas, en el que cada una lleva asociada un mapa individual del entorno (Consultar capítulo 6). En la etapa de remuestreo, aquellas partículas que tienen peores pesos son eliminadas. Las muestras con las que trabajamos son las mejores y por tanto los mapas que obtenemos tienen una calidad mayor.

7.2. Experimentos y resultados de robot real: Manfred.

Tras probar el funcionamiento de ambos algoritmos con el simulador Player/Stage se ha procedido a realizar diferentes experimentos con el robot real Manfred. Las pruebas se realizaron en el pasillo de la tercera planta del edificio Betancourt de la Universidad Carlos III de Madrid (Figura 7.9) y en uno de los laboratorios situados en la misma planta.

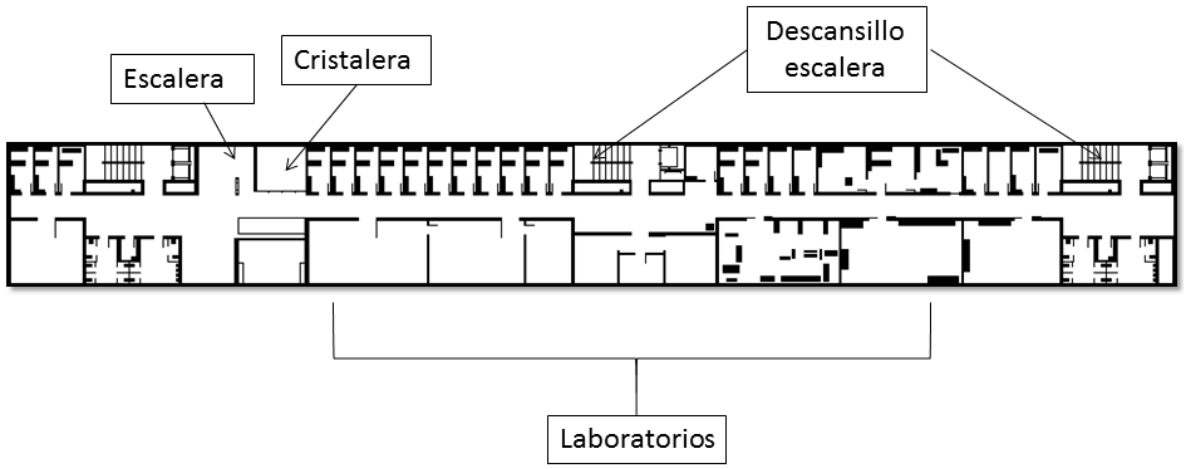


Figura 7.9: *Diseño CAD de la tercera planta del edificio Betancourt de la Universidad Carlos III de Madrid.*

La experimentación va a consistir en realizar un recorrido a lo largo del pasillo de la tercera planta del edificio Betancourt. Se partirá de una posición inicial y se regresará de nuevo a dicha posición. Lo que se pretende observar es el error que introducen los algoritmos en la estimación de la posición tras una trayectoria larga del robot, y como éstas circunstancias afectan a la construcción del mapa. A la hora de analizar los resultados tendremos que tener en cuenta que se trata de entornos dinámicos y complicados, en los que existe presencia de personas, puertas, mesas y sillas.

Los datos de odometría y láser necesarios para crear el mapa del entorno serán proporcionados por Manfred. Como se detalló en el capítulo 4, el robot está equipado con un láser PLS-220 de la casa SICK, ajustado a una resolución angular de 0.25^{o} y que abarca un ángulo de 190^{o}. El alcance utilizado para realizar los experimentos es de 7 metros y el número de medidas en cada barrido láser es de 761.

EXPERIMENTO 1

La figura 7.10, representa el mapa obtenido aplicando el algoritmo Gmapping. Se aprecia cómo a pesar de trabajar con datos reales, y una odometría pobre el modelo del entorno construido es bastante fiel a la realidad.

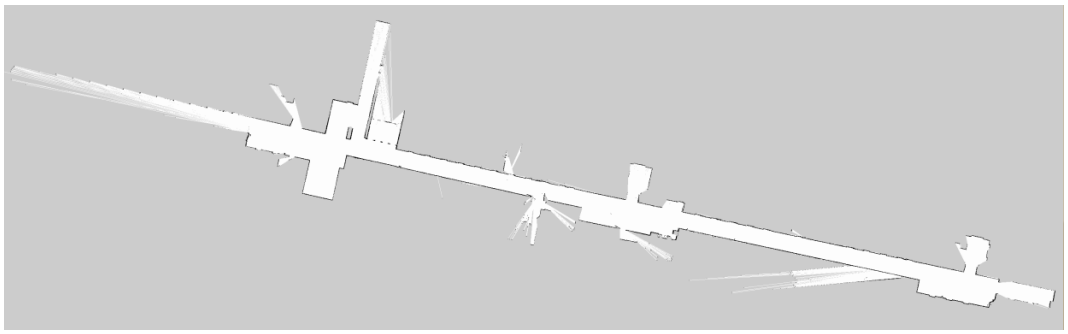


Figura 7.10: *Pasillo de la tercera planta del edificio Betancourt.*

Como comentamos al inicio de la sección, estamos trabajando en entornos dinámicos donde el estado de los objetos puede cambiar. Este efecto se puede apreciar en la representación del mapa, donde el láser ha sido capaz de apreciar aquellas estancias que se encuentran con las puertas abiertas (círculos azules). Por otro lado, los descansillos de las escaleras también han sido reconocidos (círculos verdes).

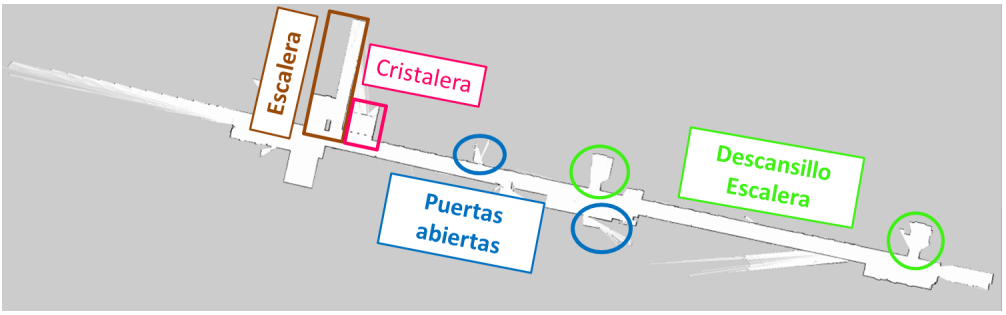


Figura 7.11: *Pasillo de la tercera planta del edificio Betancourt.*

Un efecto interesante, puede observarse en el hueco de la escalera situado en la parte izquierda de la fotografía (rectángulo marrón). En el diseño CAD de la tercera planta del edificio Betancourt se ha representado una pared, pero en la imagen que se obtiene tras el experimento nos encontramos con un haz que se extiende más allá, esto es debido a que realmente no hay pared sino una escalera.

Por último, otro detalle importante se encuentra en la zona correspondiente a las cristaleras situadas al final del pasillo (rectángulo rosa). Los cristales producen un efecto reflectante que lleva consigo que se represente en el mapa un obstáculo que no existe en la realidad. Se aprecia una imagen desdoblada del pasillo y la cristalera.

EXPERIMENTO 2

A continuación, se muestran los resultados obtenidos al desplazar al robot real Manfred a través de uno de los laboratorios del Departamento de Automática y a lo largo del pasillo del mismo área.

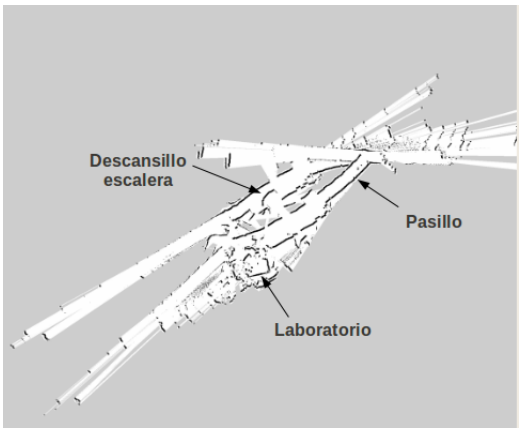


Figura 7.12: *Resultado CoreSlam.Pasillo de la tercera planta del edificio Betancourt.*

En este caso, hemos procesado los datos tanto con el algoritmo CoreSlam (Figura 7.12), como con Gmapping (Figura 7.13). Observamos que el modelo de entorno que nos ofrece CoreSlam es bastante malo, a pesar de que se puede identificar tanto el laboratorio como el pasillo, el algoritmo deja de funcionar en el trayecto de regreso al punto de partida. Esto es debido a que la odometría es mala y al tipo de suelo por el que se desplaza el robot.

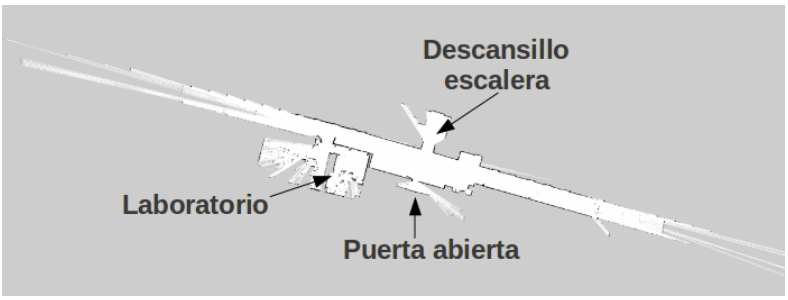


Figura 7.13: *Resultado Gmapping.Pasillo de la tercera planta del edificio Betancourt.*

Todo lo contrario ocurre en el caso de Gmapping. El algoritmo muestra unos resultados satisfactorios, siendo capaz de corregir la odometría y asociar los datos correctamente. Se puede apreciar el descansillo de la escalera y reconocer tanto el laboratorio como el despacho contiguo. Al igual que ocurría en los resultados del experimesento 1, en este caso el láser también percibe la existencia de puertas abiertas.

EXPERIMENTO 3

De nuevo volvemos a conducir a Manfred a lo largo del pasillo del edificio Betancourt, hemos querido mostrar los resultados obtenidos en esta ocasión para demostrar que todos los algoritmos pueden fallar.

Como se observa en la figura 7.14, en el camino de regreso al laboratorio el robot se desvía bruscamente de su trayectoria y a partir de este momento no es capaz de correguir su localización. Como consecuencia nos devuelve un mapa que no se corresponde con la realidad. El problema ha sido causado por las tres ruedas que dan estabilidad al robot Manfred y que evitan su vuelco. El diámetro de éstas es grande por lo que cualquier

pequeño derrape por parte de alguna de ellas tiene consecuencias catastróficas. La imagen del mapa que genera CoreSlam no se muestra debido a los malos resultados que se obtienen.

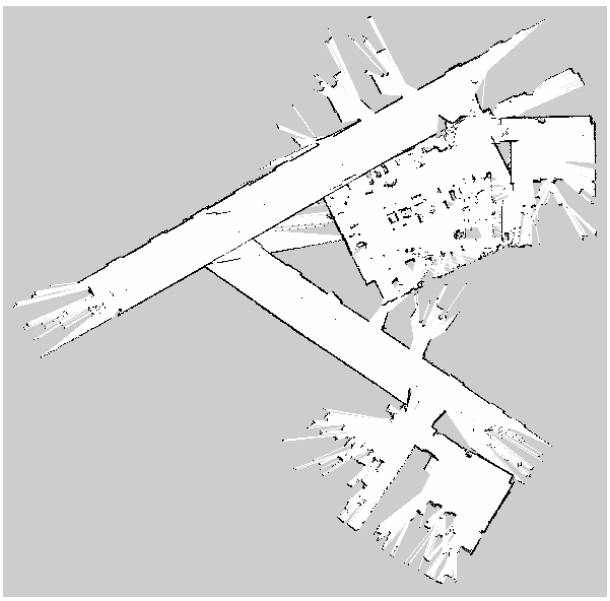


Figura 7.14: *Resultado Gmapping.Pasillo de la tercera planta del edificio Betancourt.*

EXPERIEMENTO 4

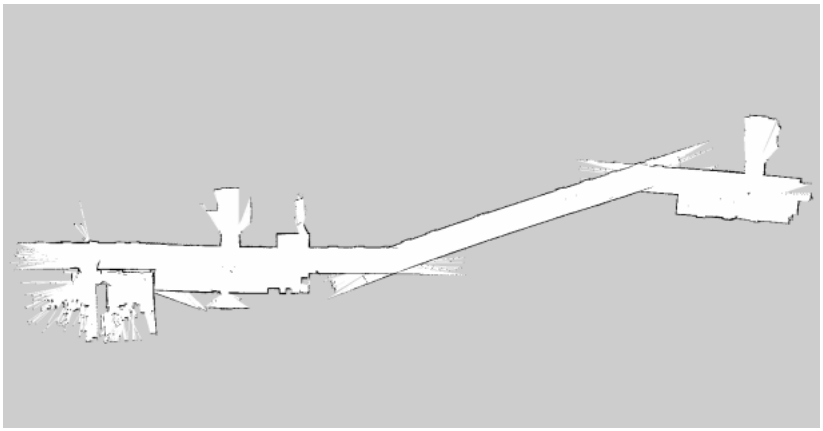


Figura 7.15: *Resultado Gmapping.Pasillo de la tercera planta del edifico Betancourt.*

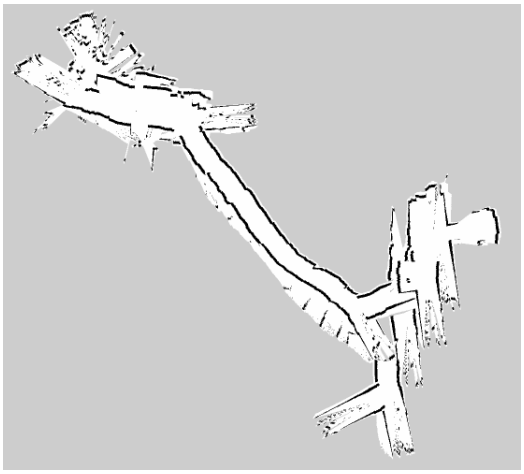


Figura 7.16: *Resultado CoreSlam.Pasillo de la tercera planta del edificio Betancourt.*

Al igual que en los experimentos anteriores, los resultados obtenidos con el algoritmo Gmapping (Figura 7.15) son mejores y más precisos que los conseguidos por CoreSlam (Figura 7.16). A pesar de esto, se puede apreciar un fallo en la trayectoria realizada por Manfred que se corresponde con un cambio en el tipo de suelo, pasando de una superficie lisa como es el mármol, a una superficie con una cierta textura. Estos cambios desequilibran a Manfred, haciendo que las ruedas encargadas de evitar su vuelco no lleven la misma dirección.

Una posible solución consistiría en cambiar las ruedas de apoyo por otras de menor diámetro, cuya superficie de contacto con el duelo fuése mínima. Por otro lado, para reducir los errores que se producen cuando el robot no es capaz de recuperar su posición, se propone volver a visitar las zonas en las que ya se ha estado.

Otro motivo por el que la representación del algoritmo Gmapping es más precisa, es que este algoritmo obtiene un mapa por cada partícula. Elimina aquellas muestras cuyo peso es más bajo, es decir, cuya representación es peor, lo que nos asegura mejores resultados. En cambio, el algoritmo CoreSlam sólo realiza un mapa, siendo éste la única solución posible.

7.3. Comparativa entre los algoritmos.

Después de realizar numerosos experimentos y trabajar tanto en entornos simulados como en entornos reales, concluimos que el algoritmo que mejores resultados obtiene es Gmapping. Especialmente cuando se trata de entornos reales y de grandes dimensiones. Por su parte, el algoritmo CoreSlam es bastante sensible a características desfavorables como dinamismos en el ambiente y objetos difíciles (sillas, mesas, cables, etc.).

El algoritmo CoreSlam falla a la hora de asociar datos y de cerrar bucles. No siendo capaz de identificar qué puntos del espacio fueron observados con anterioridad, por su parte, Gmapping sí es capaz de obtener medidas correctas en estas situaciones. Llegamos a la conclusión de que Gmapping es un algoritmo más preciso, robusto y eficiente.

Las diferencias fundamentales entre ambos algoritmos son:

a) El algoritmo de localización empleado es diferente. CoreSlam utiliza un filtro de Monte-Carlo, mientras que Gmapping emplea scan matching.

b) CoreSlam, a pesar de ser un algoritmo basado en filtro de partículas, sólo asocia un mapa a cada partícula. En cambio, el algoritmo Gmapping utiliza un filtro de partículas, en el que cada partícula construye un mapa independiente del entorno, y filtra sobre ellos calculando en cada instante el mapa con mayor probabilidad de representar la realizada observada.

Capítulo 8

Conclusiones y trabajos futuros.

Una vez descritos y probados, en los capítulos 6 y 7, los algoritmos CoreSlam y Gmapping, queremos finalizar el documento resumiendo las principales conclusiones a las que hemos llegado y estudiar hasta qué punto se han logrado los objetivos planteados al inicio del proyecto (capítulo 1). Por último, comentaremos algunos de los trabajos futuros que pueden ser desarrollados como continuidad a este trabajo.

8.1. Conclusiones.

Teniendo en cuenta los objetivos que se plantearon en el capítulo 1, que principalmente se dividen en tres bloques: aprender los conocimientos y términos de la Localización y Modelado Simultáneo, experimentar sobre simuladores y robots reales y conocer y utilizar la herramienta de programación ROS. En esta sección, comentaremos hasta qué punto se han logrado cada uno de ellos.

Cuando comenzamos el proyecto no teníamos ningún conocimiento sobre técnicas de Localización y Modelado Simultáneo, pero tras 10 meses de intenso trabajo hemos adquirido los suficientes conocimientos como para hacer frente a un problema SLAM. Para comprender las técnicas de SLAM, hemos tenido que estudiar otros conceptos relacionados como son: teoría probabilística, scan matching, occupancy grid, etc. Estos objetivos han sido sobradamente alcanzados a lo largo del desarrollo del proyecto.

El segundo de los objetivos también se ha alcanzado, ya que se ha probado el funcionamiento de los algoritmos CoreSlam y Gmapping tanto en la plataforma de simulación Player/Stage como en el robot real Manfred. Comprobando que el algoritmo Gmapping es mucho más robusto que CoreSlam. Esto es debido, a que el algoritmo de localización que emplea Gmapping es más preciso. Además, también influye el hecho de que el algoritmo CoreSlam sólo obtiene un mapa del entorno, mientras que, Gmapping asocia a cada partícula un mapa independiente.

En cuanto al tercer objetivo que nos planteamos, hemos comprendido el funcionamiento básico de ROS y hemos aprendido a utilizar muchas de las herramientas que nos ofrece. Como el visor, los ficheros de datos .bag, los archivos .launch, etc.

8.2. Trabajos futuros.

A pesar de que en la actualidad no existe un método infalible para la construcción de mapas cada vez se trabaja más activamente en este área de investigación, surguiendo innovaciones, hallazgos y resultados interesantes y novedosos día a día. Es por esto, que algunas de las líneas de trabajo futuras que proponemos son:

a) En el caso del algoritmo Gmapping, reducir el número de partículas necesarias para ahorrar espacio de memoria.

b) Aproximar mejor la distribución de probabilidad real, teniendo en cuenta el objetivo del punto a.

c) Algoritmos de remuestreo más eficaces que no eliminen hipótesis correctas.

d) Aplicar técnicas de filtros evolutivos como posible sustitución del filtro de partículas.

Apéndice A

Player/Stage

Ejemplo de fichero .world empleado por la herramienta de simulación Player/Stage

Sólo mostramos aquellos bloques que consideramos más importantes o que tienen mayor peso en la ejecución de este proyecto, aunque la herramienta Player/Stage permite simular y diseñar gran variedad de dispositivos.

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
//Definimos el telémetro láser, diseñado con un rango
máximo de 15m, cuyo campo de visión es de 180 radianes
y un total de 181 muestras en cada barrido láser.

define topurg laser
(
  range_max 15.0
  fov 180
  samples 181
  # generic model properties
  color "black"
  size [ 0.05 0.05 0.1 ]
)
```

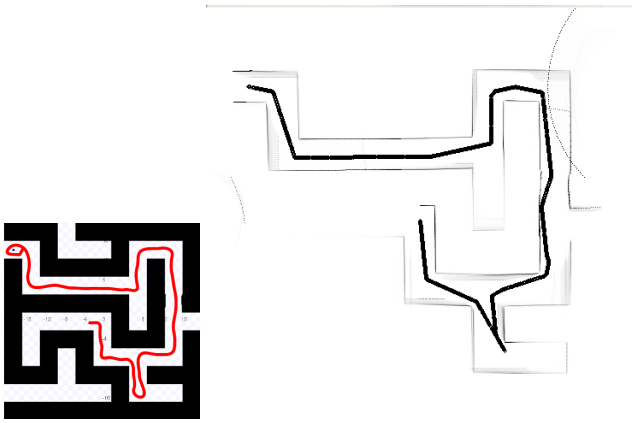
```
//Definimos la posición del robot, en este caso
no se introduce error, si se quisiese simular
ruido deberíamos añadir el término:odom_error

define erratic position
(
  size [0.35 0.35 0.25]
  origin [-0.05 0 0 0]
  gui_nose 1
  drive "diff"
  localization "gps"
  #odom_error [0.05 0.05 0.00 0.05]
  topurg(pose [ 0.050 0.000 0 0.000 ])
)

//Definimos el entorno por el que se
moverá el robot.

(
  name "maze02"
  bitmap "maze02.pgm"
  size [ 40 40 1]
  pose [ 0 0 0 0.0 ]
)
```

Apéndice B

Azumapping

Como comentamos en el capitulo 4, cuando comenzamos este proyecto no teníamos ni conocimientos sobre técnicas SLAM,ni sobre cómo trabajar con la plataforma ROS. Es por esto, que antes de llevar a cabo la implementación del módulo Gmapping nos planteamos crear un sencillo programa que nos ayudase a familiarizarnos con dicha herramienta. Permitiéndonos comprender cada uno de los conceptos que están involucrados en el paquete SLAM. Además de lo anterio, tuvimos que documentarnos a cerca de técnicas probabilísticas [37], localización, scan-matching [18], occupancy-grid [11] y SLAM [32].Un ejemplo de los resultados obtenidos pueden verse en la figura B.1

El primer paso consistió en crearnos un paquete que contuviese todos los nodos, librerías y archivos necesarios. Seguidamente escribimos el código del programa y simulamos los datos y entornos con la herramienta Player/Stage. El nodo azumapping, se suscribe mediante mensajes a los datos publicados en el topic del láser y de odometría.



(a) Mapa real (b) Mapa simulado

Figura B.1: *Mapa generado con el algoritmo azumapping.*

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
ros::init(argc, argv, "azumapping");
//Inicializamos ROS. El tercer argumento es el nombre del nodo.
ros::NodeHandle n;

ros::Subscriber odom_sub =
 n.subscribe("odom", 50, odomCallback);// Se suscribe al
topic "odom" .

ros::Subscriber base_scan =
 n.subscribe("base_scan", 50, base_scanCallback);// Se suscribe
al topic "base_scan"
```

A medida que se reciben los valores de láser y odometría se va construyendo el mapa del entorno.

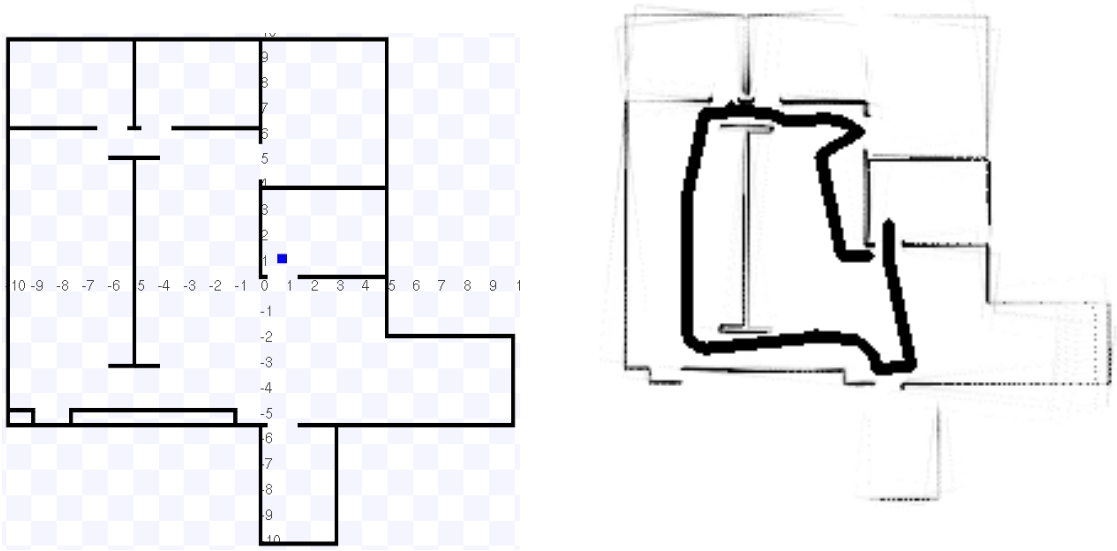
```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
//Definimos las funciones que serán llamadas cuando
 un mensaje llegue al correspondiente topic.
void base_scanCallback(const sensor_msgs::LaserScan::ConstPtr& base_scan)
ROS_INFO("Procesando laser")}

//Función que es llamada cuando un nuevo mensaje llega al topic odom.
void odomCallback(const nav_msgs::Odometry::ConstPtr& odom)
ROS_INFO("odometria: %.1f, %.1f, %.1f", x, y, w);
//obtenemos medidas dadas por la odometría
```

Dicha imagen la construimos usando funciones de programación de la librería OpenCV (consultar capítulo 4).

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
//Creamos una imagen
mapimg = cvCreateImage(cvSize(500, 500), IPL_DEPTH_8U, 1);

//pintamos un circulo=robot
cvCircle(mapimg, cvPoint(x * 10 + 250, y * 10 + 250), 2,
cvScalar(0, 255, 0), 1);
 //Mostrar la imagen en la ventana
cvShowImage("azumapping", mapimg);
```

En esta etapa del proyecto, también nos familiarizamos con el concepto launch, explicado en el capitulo 5, que permite lanzar en un único terminal todos los nodos involucrados. A parte de esto, practicamos con la instrucción rosrunbag que nos permitió grabar los datos que íbamos reibiendo en un archivo .bag (consultar capítulo 5), para poder reproducirlos más tarde.

A continuación, mostramos una imágen (Figura B.2) de uno de los experimentos llevados a cabo. A pesar de ser un mapping muy rudimentario, puede observarse cómo se va construyendo el mapa a medida que se van recibiendo los datos de láser y odometría.



(a) Mapa real (b) Mapa simulado

Figura B.2: *Mapa generado con el algoritmo azumapping.*

Apéndice C

Launch

Todo algoritmo de Localizacion y Modelado Simultáneo requiere que una gran cantidad de nodos se ejecuten al mismo tiempo, y por tanto, trabajar con un alto número de terminales en los que ejecutar las órdenes necesarias. Para facilitar y simplificar esta labor, ROS dispone de una herramienta llamada launch que nos permité lanzar al maestro de la aplicación de manera inmediata, y al mismo tiempo poner en funcionamiento todos los nodos necesarios.

Añadimos los ficheros .launch que hemos creado en este proyecto. Sólo adjuntamos los correspondientes al algoritmo CoreSlam, en el caso del algoritmo Gmapping las instrucciones son la mismas con la diferencia que en lugar de ejecutar el paquete CoreSlam, se hace una llamada al paquete Gmapping. En primer lugar, se muestra el fichero con el que grabamos los datos y seguidamente el fichero que nos permite reproducir esos mismos datos.

Launch que nos permite grabar datos

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
<launch>

   <arg name="nombre_bag" default="cualquiera" />
   <param name="use_sim_time" value="true"/>
// Nodo que lanza el paquete de simulación, el fichero.world
 variará en función del entorno que querramos representar.
  <node pkg="stage" name="simulator" type="stageros"
args="$(find stage)/world/willow-erraticMio.world"/>
  //Nodo del algoritmo Coreslam
   <node pkg="coreslam" name="coreslam" type="slam_coreslam"
 args="scan:=base_scan"/>
   //Nodo de teleoperación
   <node pkg="teleop_base" name="teleop" type="teleop_base_keyboard"
 args="base_controller/command:=cmd_vel" />
   // Nodo para lanzar la herramienta de visualización.
   <node pkg="rviz" name="rviz" type="rviz"/>
   // Nodo que permite grabar los datos publicados en un archivo.bag
   <node pkg="rosbag" name="nodo_rosbag" type="record"
args="-O /home/azucena/Dropbox/Manfred_v2/Azucena/MyROS/coreslam/bag/
$(arg nombre_bag) base_scan
 odom tf"/>
</launch>
```

Launch que nos permite reproducir datos

```
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
<launch>
<arg name="nombre_bag" default="cualquiera" />
<param name="use_sim_time" value="true"/>
//Nodo del algoritmo Coreslam
<node pkg="coreslam" name="coreslam" type="slam_coreslam"
 args="scan:=base_scan"/>
Nodo de la herramienta de visualización.
<node pkg="rviz" name="rviz" type="rviz"/>
//Nodo que permite reproducir los datos guardados en un archivo .bag
<node pkg="rosbag" type="play" name="nodo_rosbag"
args="/home/azucena/Dropbox/Manfred_v2/Azucena/MyROS/coreslam/bag/
$(arg nombre_bag)" />
</launch>
```

Apéndice D

Planificación. Diagrama de Gantt

En la figura D.1 se muestra el diagrama de Gantt de la planificación del proyecto. Ha tenido una duración aproximada de 10 meses, incluyendo la fase inicial de formación. El comienzo tuvo lugar en Noviembre de 2010 y la finalización se establece a finales de octubre del 2011. Durante el período comprendido entre el mes de noviembre y el mes de julio, el proyecto se ha compaginado con la finalización del último año de carrera.

A continuación, describimos las fases más importantes del proyecto. Debemos decir que las etapas de documentación y aprendizaje han abarcado los 10 meses que ha durado el proyecto, ya que la plataforma ROS se actualiza a un ritmo vertiginoso y el estudio de técnicas SLAM es muy amplio y todavía queda mucho por investigar.

Sistema Operativo Linux Instalación y configuración del sSistema Operativo Linux en el PC, y posterior aprendizaje de los conceptos y comandos fundamentales.

C++ Repaso de los conceptos básicos, y aprendizaje de conceptos nuevos no estudiados a lo largo de la carrera.

Investigación y documentación Estudio de los conceptos básicos de SLAM (Simultaneous Localization And Mapping)

ROS Instalación de ROS y estudio de la herramienta. Durante un mes estuvimos familiarizándonos con la que sería nuestra herramienta de trabajo, para ello realizamos los tutoriales disponibles en la página web de ROS. Durante los meses que ha dura-

do el proyecto, se han ido ampliado los conocimientos sobre ROS y se ha aprendido a trabajar con las diferentes herramientas y recursos que nos proporciona.

Documentación La labor de documentación ha sido muy intensa, sobre todo en los meses iniciales.

Simulador Player/Stage Aprendizaje de los conceptos de Player/Stage realizando sencillos experimentos para afianzar conceptos.

Módulo Gmapping Instalación y compilación del paquete Slam y Carmen.

Estado del arte Se estudia la literatura escrita hasta el momento sobre técnicas SLAM, así como todos aquellos conceptos necesarios para su compresión, tales como scan matching, occupancy grid, etc. También ha sido necesario aprender ciertos conceptos probabilísticos [37].

Simulación Desarrollamos el módulo azummapping, el objetivo era afianzar los conceptos necesarios para trabajar con la herramienta ROS y entender cada uno de los pasos realizados en los algoritmos de Localización y Modelado Simultáneo. Consultamos los manuales de ROS y aprendemos cómo se realiza la comunicación entre nodos, creamos módulos de láser, odometría y transformación.

Algoritmo CoreSlam Instalación y estudió de la documentación acerca del algoritmo tinySlam. Se realizan experimentos simulados con Player/Stage.

Algoritmo Gmapping Implementación del módulo Gmapping, primeras pruebas y resultados.

Manfred Primer contacto con Manfred. Cuando se trata de implementar el algoritmo en el robot real Manfred surgen alguas dificultades. Debemos revisar los nombres de los nodos y topics para comprobar que la comunicación entre módulos se realiza correctamente.

Experimentos III Pruebas definitivas en entornos simulados y en Manfred. Se realizan los experimentos en la tercera planta del edificio Betancourt de la Universidad Carlos III de Madrid.

Memoria Instalar y aprender el funcionamiento del editor de texto Latex. Se redacta el documento escrito y la presentación power point.

	Nombre de la tarea	P4 2010			P1 2011			P2 2011			P3 2011			P4 2011		
		Oct	Nov	Dec	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1	**Instalación**															
2	Instalar sistema Operativo Linux															
3	Instalar Carmen															
4	Instalar ROS															
5	Compilar módulo Gmapping															
6	**Formación**															
7	Aprendizaje básico C++															
8	Aprendizaje ROS: tutoriales															
9	Aprendizaje herramientas ROS															
10	Aprendizaje Player/Stage															
11	Aprendizaje Player/Stage															
12	Aprendizaje Player/Stage															
13	**Documentación**															
14	Búsqueda y lectura información SLAM															
15	Búsqueda información SLAM															
16	Consulta documentacion OpenSlam,															
17	Consulta documentación ROS															
18	Consulta documentación Player/Stage															
19	**Estado del arte**															
20	Lectura libro Probabilistic Robotics															
21	Papers SLAM															
22	Tesis Cyrill Stachniss															
23	Scan-matching															
24	Ocupancy-grid															
25	**Programación I**															
26	Programa azumapping															

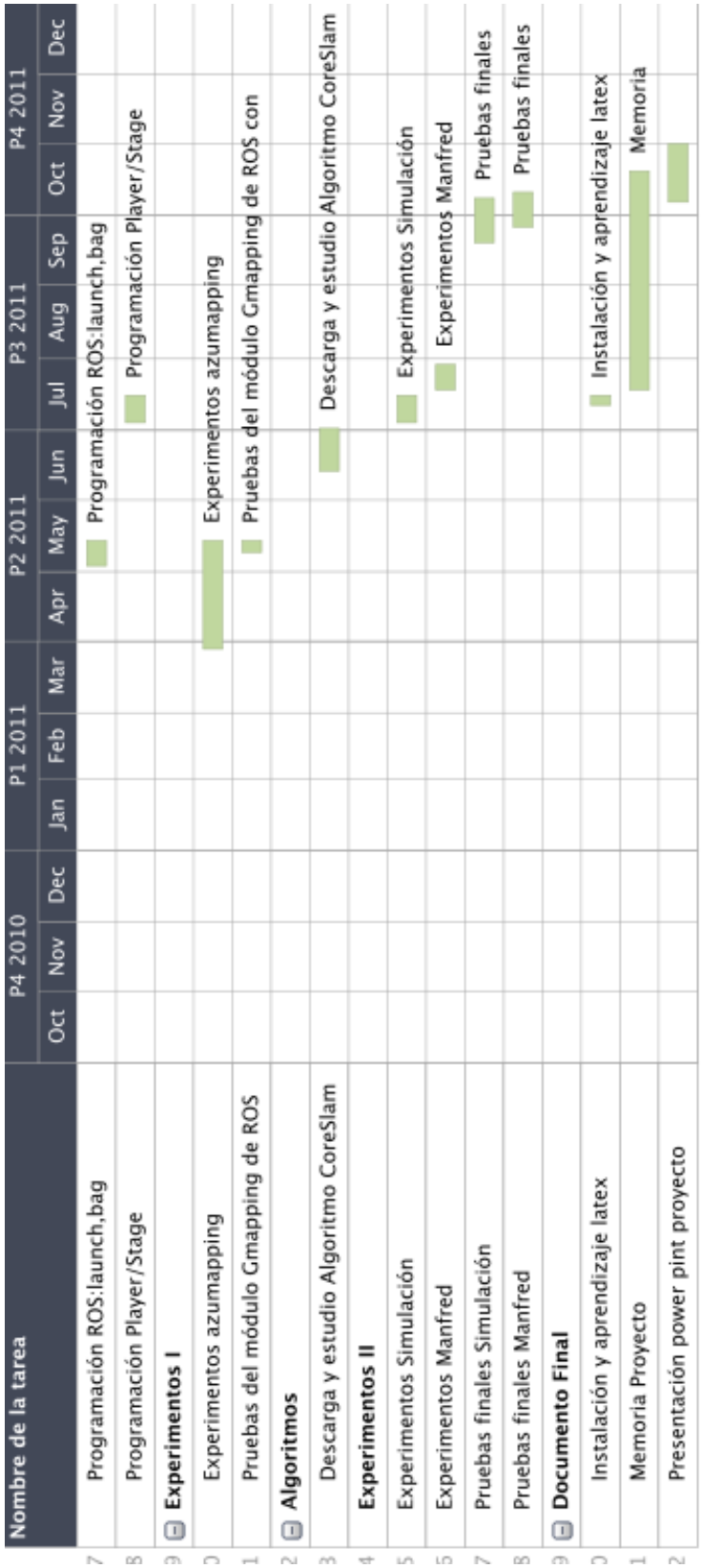


Figura D.1: *Planificación. Diagrama de Gantt.*

Bibliografía

[1] T. Bailey. Mobile robot localisation and mapping in extensive outdoor environments. 2002.

[2] J. Borenstein, HR Everett, and L. Feng. Where am i? sensors and methods for mobile robot positioning. *University of Michigan*, 119:120, 1996.

[3] J.W. Branco, J.B. Gomez, and P. Flavio. Reconstrucción tridimensional a partir de imágenes de rango. *Revista Iberoamericana de inteligencia Artificial*, 8(23), 2004.

[4] Algoritmo Bresenham. http://en.wikipedia.org/wiki/bresenham's line algorithm, Accedido en Septiembre 2011.

[5] Algoritmo Bresenham. http://www.falloutsoftware.com/tutorials/dd/dd4.htm, Accedido en Septiembre 2011.

[6] E. Charniak. Bayesian networks without tears. *AI magazine*, 12(4):50, 1991.

[7] R. Diankov and J. Kuffner. Openrave: A planning architecture for autonomous robotics. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-08-34*, 2008.

[8] A. Diosi and L. Kleeman. Laser scan matching in polar coordinates with application to slam. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3317–3322. IEEE, 2005.

[9] H.F. Durrant-Whyte. Uncertain geometry in robotics. *Robotics and Automation, IEEE Journal of*, 4(1):23–31, 1988.

[10] A. Elfes. Sonar-based real-world mapping and navigation. *Robotics and Automation, IEEE Journal of*, 3(3):249–265, 1987.

[11] A. Elfes. Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[12] G. Grisettiyz, C. Stachniss, and W. Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 2443–2448. IEEE, 2005.

[13] D. Hhhnel, W. Burgard, B. Wegbreit, and S. Thrun. Towards lazy data association in slam. In *Proceedings of the 11th International Symposium of Robotics Research (ISRR)*.

[14] J.P. Bandera C. Urdiales F. Sandoval J.M. Pérez. Agente autónomo de bajo coste para la exploración de conductos teleoperado mediante realidad virtual. *I Seminario Nacional Hispabot (HISPABOT 03)*, Abril, 2003.

[15] D. Kaehler and G. Bradski. Learning opencv. *Farnham: O'Reilly*, 2008.

[16] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. *Journal of Intelligent and Robotic Systems*, 18(3):249–275, 1997.

[17] S. Maskell and N. Gordon. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *In IEEE Colloquium on Tracking*, 2002.

[18] J. Minguez, F. Lamiraux, and L. Montesano. Metric-based scan matching algorithms for mobile robot displacement estimation. In *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, volume 4, page 3557. Citeseer, 2005.

[19] L. Montesano, J. Minguez, and L. Montano. Probabilistic scan matching for motion estimation in unstructured environments. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3499–3504. IEEE, 2005.

[20] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 116–121. IEEE, 1985.

[21] Página oficial de OpenSlam. http://openslam.org/, Accedido en Septiembre 2011.

[22] Página oficial de ORCA. http://orca-robotics.sourceforge.net/, Accedido en Septiembre 2011.

[23] Player/Stage-Project. http://playerstage.sourceforge.net/, Junio, 2005.

[24] M. Quigley, E. Berger, A.Y. Ng, et al. Stair: Hardware and software architecture. In *AAAI 2007 Robotics Workshop, Vancouver, BC*, 2007.

[25] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[26] Radish. http://radish.sourceforge.net/, Accedido en Septiembre 2011.

[27] Stanford Artificial Intelligence Robot. http://stair.stanford.edu, Accedido en Septiembre 2011.

[28] Microsoft Robotics. http://microsoft.com/robotics, Accedido en Septiembre 2011.

[29] F.J.C. Sierra. *Enciclopedia del lenguaje C++*. Ra-Ma, 2003.

[30] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. *Autonomous robot vehicles*, 1:167–193, 1990.

[31] R.C. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *The international journal of Robotics Research*, 5(4):56, 1986.

[32] C. Stachniss. Exploration and mapping with mobile robots. *Universitt Freiburg*, 2006.

[33] S. Thrun. Particle filters in robotics. In *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, volume 1. Citeseer, 2002.

[34] S. Thrun. Particle filters in robotics. In *Proceedings of the 17th Annual Conference on Uncertainty in AI (UAI)*, volume 1. Citeseer, 2002.

[35] S. Thrun. Robot mapping: A survey. *Exploring Artificial Intelligence in the New Millenium, Morgan Kaufmann*, 2002.

[36] S. Thrun. Robot mapping: A survey. *Exploring Artificial Intelligence in the New Millenium, Morgan Kaufmann*, 2002.

[37] S. Thrun, W. Burgard, and D. Fox. Probabilistic robotics. 2005.

[38] R.T. Vaughan and B.P. Gerkey. Really reusable robot code and the player/stage project. *Software Engineering for Experimental Robotics. Springer*, 2007.

[39] Dirección web de la plataforma ROS. http://www.ros.org/, Accedido en Septiembre 2011.

[40] G. Welch and G. Bishop. An introduction to the kalman filter. *University of North Carolina at Chapel Hill, Chapel Hill, NC*, 7(1), 1995.